## WELCOME TO THE VBPJ TECHNICAL TIPS SUPPLEMENT!

**We survey our readers often, and one of the most consistently high-scoring replies to our surveys is, "Give us tips, tricks, and workarounds for making our VB programs smaller, faster, and more powerful!" Actually, we dedicate the features and columns of each issue to fulfilling this request because it's the core of our charter.**

**However, when reviewing our latest reader survey, we decided, "Because our readers want tips and tricks, why not put together an entire supplement that's nothing but insights, nuggets, and workarounds for improving VB programs and the programming process itself?" *Violá*. We contacted programmers far and wide, using CompuServe, the Internet, e-mail, and old-fashioned telephone calls—beating the bushes for every tip we could find.**

**And find them we did. We received far more than we could publish. We mused over them, sent them to members of the *VBPJ* Technical Review Board for critique and analysis (our Review Board members also included their favorites). We whittled and hacked until we had our favorite 99 tips (plus 11 bonus tips we just couldn't leave out).**

**If these tips don't improve your VB programs and make you a better VB programmer, nothing will. Enjoy.**

## GET THE LATEST VB-RELATED FILES

You can get the latest Microsoft files for VB as well as other Microsoft products on CompuServe in the Microsoft Libraries Forum (GO MSL). The MSL includes updated help compilers, setup kits, updated VBRUNX00.EXE files, database files (such as the compatibility layer that will allow you to use Access 2.0 files), and many others. You can also download the latest Visual Basic Knowledge Base file, which includes invaluable information about programming in VB. (The Knowledge Base is available as text and help files. The help file is more useful but takes longer to download.)

Four other CompuServe areas are of interest to VB programmers. The Windows Component Forums (COMPA and COMPB) are vendor forums where you can get customer support (and updated files) from a wide range of VBX/OCX makers. The Microsoft Basic (MSBASIC) Forum and this magazine's VBPJFO Forum have areas that help with your development questions and provide VB-related files. All four forums include a wealth of shareware, source code samples, and information about programming tools.

On the Internet, check out the USENET newsgroup comp.lang.basic.visual, as well as related newsgroups such as comp.lang.basic.visual.database and comp.lang.basic.visual.misc. These areas aren't as well attended (or organized) as the four CompuServe forums, but they contain a great deal of useful information. (CompuServe users can get to Internet sites by typing GO INTERNET.) Make sure to download the latest FAQ (or Frequently Asked Questions) list from the anonymous ftp archive site rtfm.mit.edu. All parts of the VB FAQ are in the directory pub/usenet/comp.lang.basic.visual.

Another source of VB tips is Dave McCarter's "Visual Basic Tips and Tricks" help file. It's called VBTTXX.ZIP (where "XX" is the version number). It's been uploaded to a number of sources, including the VBPJFO forum.

**—*VBPJ* Staff**

## USING DATABASE TRANSACTIONS

When you first start using the VB database functions, you might wonder why they call its engine JET—it seems to *not* fly. Using database transactions (search the online help for BeginTrans) might at first seem like a technique you wouldn't want to use, but it can significantly speed up your database manipulations. An easy way to get up to speed with this functionality is to put a BeginTrans statement just before your dynaset update code and a CommitTrans statement just after it.

**—Michiel de Bruijn**

## CLOSE VB BEFORE COMPILING

When you're finished tinkering with your apps, close and restart VB before making the final EXE. This simple action can reduce the size of your EXE by 10 to 30 percent (many professional programmers also recommend restarting Windows before building an EXE). If you don't close and restart VB, your EXE may contain some garbage: VB doesn't fully clean up all the data structures or variables you used during development.

Restarting VB also safeguards against some mysterious GPFs. If you have an app that runs fine in the development environment but GPFs when it's run as an EXE, try closing and restarting. Another option is to compile from the "command line." To do so from either Program Manager or File Manager, select Run from the File menu, and enter:

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

**—Patrick O'Brien and Karl Peterson**



HOME

## USING RELATIVE REFERENCES FOR FLEXIBLE DIRECTORY STRUCTURES

You can use relative references to specify file paths in your MAK file. For example: Path="..\SETUP\DISK1." When you use relative references, you can move entire directory structures in File Manager without having to rebuild your MAK files.

**—Craig Goren**

## ACTIVATING THE PREVIOUS INSTANCE OF YOUR APP

There are many times when you may not want to allow more than one instance of your application to be launched. In such cases, your program needs to determine if an instance is already running and, if so, activate and close the previous instance. If no previous instance is detected, the program continues normally.

The AnotherInstance function determines if the program is already running. If it is, the previous instance is activated and the function returns True. If no previous instance is running, the function returns False. You should call this function when your program starts, preferably from Sub Main. If it returns True, then the program should terminate:

```
'Activates the previous instance
Function AnotherInstance ()
   Dim appTitle$

   If App.PrevInstance Then
      appTitle$ = App.Title
      'Get our application name
      App.Title = "~!@#$%^&"
      'Set new instance name to unlikely value
      AppActivate appTitle$
      'Activate previous instance
      AnotherInstance = True
   Else
      AnotherInstance = False
   End If
End If
```

AnotherInstance works by checking the PrevInstance property of the App object. If PrevInstance is not zero, then the program is already running. In this case, the function activates the previous instance using AppActivate. Note that AppActivate activates the application with the specified window caption. To prevent AppActivate from activating the current instance, the Title property must be set to a value that is not likely to be the actual caption. Note that this also means that this technique will not work if your application modifies the window caption (Title).

**—Jonathan Wood**

## CREATING DISABLED OR ETCHED ICONS

Here's how to create an etched or disabled icon using an icon editor:

1. Change all filler color to light gray.
2. Change the outline to dark gray (usually from black).
3. Change every dark gray pixel to white if the pixel to its south east is light gray.

**—Craig Goren**

## THE MOVE METHOD IS FASTER

To move a control or form to a new location, you can set the Left and Top properties to new values:

```
frmCustomer.Left = frmCustomer.Left + 100
frmCustomer.Top = frmCustomer.Top + 50
```

Using the Move method, however, is about 40 percent faster:

```
frmCustomer.Move frmCustomer.Left + 100, frmCustomer._
   Top + 50
```

**—Paul D. Sherriff/*Visual Basic Power Guides***

## DEBUGGING EVENTS WHEN THEY'RE ACTING FUNNY

Use Debug.Print "Entering event ..." without breakpoints to trace events if they seem to act funny. Don't use break points with MsgBoxes—they can alter the event sequence.

**—Craig Goren**

## COMMENTS DON'T INCREASE EXE SIZE

VB strips comments out in the final .EXE file, so use as many comments as you need.

**—Paul D. Sherriff/*Visual Basic Power Guides***

## DEVELOP UNDER WINDOWS NT

Why not develop under Windows NT 3.5? You can crash as often as necessary <g>, and just double-click to restart VB, with virtually no need to reboot. Remember to delete temporary files created by VB (~VB*.* files in the TEMP directory) before restarting it (in either Windows or Windows NT).

Another advantage to developing under NT is that you know it will work for the users—and you know there will be a few, at least—who run that OS. Be sure to test the final application under Windows 3.*x*, however. You'll find few incompatibilities, compared to apps developed in Windows running in NT, but they can occur, especially as you become accustomed to unlimited system resources.

**—Karl E. Peterson**

## HELP SEARCH ON THE MENU BAR

Ever wanted to add a "Search for Help on..." item in the menu bar? Here's the secret:

```
'Declares and Constant for Help Menu
Declare Function WinHelp% Lib "User" _
   (ByVal hWnd%, ByVal HelpFile$, ByVal helpcode%, _
      ByVal helpdata&)
Declare Function WinHelpByString% Lib "User" _
   Alias "WinHelp" (ByVal hWnd%, ByVal HelpFile$, _
      ByVal helpcode%, ByVal helpdata$)
Const HELP_PARTIALKEY = &H105
'call the search engine in winhelp

Sub mnuSEARCH_Click ( )
   Dim R%
   R% = WinHelpByString(Me.hWnd, App.HelpFile, _
      HELP_PARTIALKEY, " ")
End Sub
```

**—Dr. B. Leckett**

## EMULATING OVERSTRIKE MODE IN TEXT BOXES

Windows text boxes always work in insert mode and don't provide an overstrike mode. However, overstrike mode can easily be emulated as shown here:

```
Sub Text1_KeyPress (KeyAscii As Integer)
   If KeyAscii >= 32 Then
      If Text1.SelLength = 0 Then
         If Text1.SelStart < Len(Text1) Then
            Text1.SelLength = 1
         End If
      End If
   End If
End Sub
```

If a key is typed when no text is selected, the code selects the current character by setting the SelLength property to 1, causing it to be overwritten by the key typed. If the cursor is already at the end of the text, setting the SelLength property to 1 is silently ignored. Several conditions are tested to ensure that a control character (such as Tab or Enter) was not pressed, that the user hasn't already highlighted text to be replaced, and that the caret is not at the end of the existing text.

**—Jonathan Wood and Karl Peterson**

## COOL COMMANDS FOR YOUR HELP MENU

Most Windows applications include the following commands in the help menu:

```
Contents
Search For Help On...
How To Use Help
```

Although VB doesn't provide direct support for these commands, they can easily be added to your VB applications by accessing the Windows API directly. The required declarations are listed here and should appear in one of your application's BAS files:

```
'Function declaration
Declare Function WinHelp Lib "User" _
   (ByVal hWnd As Integer, ByVal lpHelpFile _
      As String, ByVal wCommand As Integer, _
   ByVal dwData As Any) As Integer

'Global constants
Global Const HELP_QUIT = 2
Global Const HELP_INDEX = 3
Global Const HELP_HELPONHELP = 4
Global Const HELP_PARTIALKEY = &H105
```

Add the new menu items and name them mnuHelpContents, mnuHelpSearch and mnuHelpHowToUse, respectively. The handlers for each of these commands should look like this (Change the first argument to WinHelp (Form1.hWnd) to reference the main form in your application and set App.HelpFile to your application's help file when your application starts):



HOME

```
'Contents command
Sub mnuHelpContents_Click ()
   Dim i As Integer
   i = WinHelp(Form1.hWnd, App.HelpFile, _
      HELP_INDEX, 0&)
End Sub

'Search For Help On... command
Sub mnuHelpSearch_Click ()
   Dim i As Integer
   i = WinHelp(Form1.hWnd, App.HelpFile, _
      HELP_PARTIALKEY, "")
End Sub

'How To Use Help command
Sub mnuHelpHowToUse_Click ()
   Dim i As Integer
   i = WinHelp(Form1.hWnd, App.HelpFile, _
      HELP_HELPONHELP, 0&)
End Sub
```

To make sure Windows Help unloads when your application terminates, place this code in your main form's Unload event:

```
'Unload WINHELP.EXE
Sub Form_Unload (Cancel As Integer)
   Dim i As Integer
   i = WinHelp(frmMain.hWnd, App.HelpFile, _
      HELP_QUIT, 0&)
End Sub
```

**—Jonathan Wood**

## TEST FOR FALSE INSTEAD OF ZERO

If you're checking a numeric value against 0, one option is to use the "<>" operator:

```
If iNumber <> 0 Then
   ...
End If
```

It is faster, however, to check the variable with an If statement.

```
If iNumber Then
   ...
End If
```

These two statements are equivalent, but the second example will run faster, albeit with some loss of readability.

**—Paul D. Sherriff/*Visual Basic Power Guides***

## RIGHT-JUSTIFY MENU ITEMS

Menu items you add to your VB applications will be left justified by default. If you wish to have one or more of your menus right-justified, you can change it at run time. The trick is to add a backspace character to the caption of your menu. Because the backspace character is a Chr$(8), it cannot be entered in design mode. In the Form_Load() or other initialization routine, add this code:

```
Sub Form_Load ()
   mnuHelp.Caption = Chr$(8) & mnuHelp.Caption
End Sub
```

This code right-justifies the mnuHelp menu item. Right-justified menu items are not part of the GUI design standard set forth by Microsoft, however, so you should avoid them unless you have a good reason for using that style.

**—Paul D. Sherriff/*Visual Basic Power Guides***

## USING AUTOTAB WITH A SET MAXLENGTH

To add AutoTab to text boxes that have a set MaxLength, add this code to the text box Change event:

```
Sub Text1_Change ()
   If Len(Text1) = Text1.MaxLength Then
      SendKeys "{Tab}"
   End If
End Sub
```

This saves the user one keystroke when moving from one field to the next. It's best not to intersperse this behavior, but instead to use it on an all-or-nothing basis to prevent massive user confusion.

**—Karl E. Peterson**

## USING ALT+F4

Pressing Alt and F4 will close any window with a control box. Because you can't choose Alt and F4 as a shortcut key combination from the menu designer, you must place it as part of the caption property if you want to tell the user that this will close the window.

**—Gary Cornell**

## PROGRAMMATICALLY SELECTING MULTIPLE ROWS IN A DATA GRID

To select multiple, nonconsecutive rows (a tagged list) in Sheridan Software's DataGrid (which is a part of its Data Widgets package), first set the SelectionTypeRow property of the DataGrid to TagList. For each row you want to select, set the EvalRowNumber property to the row number and the EvalRowIsSelected property to TRUE. This code illustrates how to select odd rows in a DataGrid using this approach:

```
Sub SelectSomeRows()
   Dim i As Integer

   For i = 1 To 10 Step 2
      SSDataGrid1.EvalRowNumber = i
      SSDataGrid1.EvalRowIsSelected = True
   Next i
End Sub
```

**—Sheridan Software Tech Support**

## TESTING FOR A GIVEN FILE

There are lots of ways to test for whether a given file exists. The most common method is to use the Dir$ function. If Dir$ returns a null string, then the file doesn't exist. Couldn't be simpler, right? If an illegal filespec is tested, an error occurs. This function *depends* on an error to detect file existence, and rather than using Dir$, it uses Name.

```
Function Exists (ByVal FileSpec$)

   Dim TestSpec$
   Exists = False
   FileSpec = Trim(FileSpec)

   If Right$(FileSpec$, 1) = "\" Or _
      Right$(FileSpec$, 2) = "\." Or _
      Right$(FileSpec$, 3) = "\.." Then
      'Obviously passed a directory not a file!
      'Avoid *MAJOR BUG* with Name function by not
      'trying to rename the root directory.
      Exit Function
   End If

   On Local Error Resume Next
   Name FileSpec As FileSpec

   Select Case Err
      Case 5  'Illegal Function Call
      Case 53 'File Not Found
      Case 58 'File Already Exists (Maybe!)
         TestSpec = Dir$(FileSpec)
         If Len(TestSpec) Then Exists = True
      Case 64 'Bad Filename
      Case 68 'Device Unavailable
      Case 74 'Can't rename with different drive
      Case 71 'Disk Not Ready
      Case 75 'Path/File Access Error
      Case 76 'Path Not Found
      Case Else
         MsgBox "Unexpected Error " & Err & " _
         in Exists(): " & Error$
   End Select
End Function
```

Trying to rename a file to its current name generates error 58 — File Already Exists. If the file doesn't exist, you get error 53 — File Not Found. These errors really tell you something, don't they? There are a number of other possible errors, however, each revealing something about the filespec. The Exists function lists all the errors and their causes that I've been able to track down. Because a directory name generates the same error as a file when attempting to rename it to itself, I call Dir$ into action to confirm that a file and not a directory was passed as the filespec.

As presented, Exists is an experimental function. You'd obviously want to trim it down if you want only the final answer. By modifying this function slightly, you could also return the cause of the error, which could help you help the user correct the situation. Also, by passing many different filespecs, you might discover new ways your users could devise to break your program.

**—Karl E. Peterson**

## A HANDY UTILITY FOR WORKING WITH INI FILES

If you work with INI files, save yourself hours of work and download the INIFILE.BAS module I've uploaded to the *VBPJ* and MSBASIC Forums on CompuServe (file KPINI.ZIP). There's no charge for it, and I'm available on-line to answer questions about it (*Peterson is the sysop of the 32-Bit Bucket on the* VBPJ *Forum—Ed.*). KPINI includes more than 40 routines that can do just about anything imaginable to or with an INI file. A series of special functions deals with the oddity that is SYSTEM.INI, such as determining if a driver is loaded or even retrieving a list of all drivers. You'll find functions for retrieving all the sections with a file, all the entries within a section, erasing an entry or an entire section, and many, many other things. All functions are implemented for both WIN.INI and PRIVATE.INIs. The module can be added to any existing project. Enjoy!

**—Karl E. Peterson**



HOME

## UNDO CHANGES IN TEXT BOXES

Most professional applications have an Undo menu option under the Edit menu. If you want to have an undo feature for every text box, you may think you need to create a variable to hold the old value for every field. Fortunately, Windows takes care of this for you. With a call to the Windows API function SendMessage(), you can have the old value put back into the text box. Create a menu item called Undo and add this code:

```
Sub mnuUndo_Click ()
   Call TextUndo(Screen.ActiveControl)
End Sub

TextUndo() is responsible for performing the undo:

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

Sub TextUndo (ctlControl As Control)
   Dim lReturn As Long

   Const EM_UNDO = &H417
   If TypeOf Screen.ActiveControl Is TextBox Then
      lReturn = SendMessage(ctlControl.hWnd, _
         EM_UNDO, 0, 0&)
   End If
End Sub
```

Because some third-party controls also support the undo message, place TextUndo( ) in a separate routine to make it easy to change this one function by adding the Class name of the third-party control. If you do, there will be no reason to change all the calls to the SendMessage( ) function.

**—Paul D. Sherriff/*Visual Basic Power Guide***

## AUTOMATICALLY REPAIRING CORRUPT DATABASES

When you're using VB's built-in database functionality, you're likely to get a corrupted database sooner or later. Use this code to open your databases and automatically repair any corrupt ones:

```
On Local Error Resume Next

DidRetry% = False
Db$="My-Db.MDB"

DoOpenDatabase:
Err = 0
Set SomeDb = OpenDatabase(Db$)

If Err = 3043 Then
   'The dreaded "Disk or Network Error" ...
   MsgBox "Database Engine error. Please restart _
      Windows and this application", 16
   End
ElseIf Err = 3049 Then
   'Database corrupted
   If Not DidRetry% Then
      'Try to repair database ...
      RepairDatabase Db$
      DidRetry% = True
      GoTo DoOpenDatabase
   Else
      MsgBox "Database repair failed. Please _
         contact Tech Support", 16
      End
   End If
End If
```

**—Michiel de Bruijn**

## GRAPHICS IN MDI FORMS

To use a bitmap or other graphics in the client space of MDI forms, the only trick is getting the hWnd for that window so that you can use GDI functions on it. The key is understanding that the client space is the first child (in Windows-speak, not MDI-speak) of the MDI form.

Get the handle via the GetWindow API call with the GW_CHILD constant. You create a MDIForm_Paint event with a subclassing control such as MsgHook from *Visual Basic How To, Second Edition* by Zane Thomas, Robert Arnson, and Mitchell Waite (Waite Group Press). When the WM_PAINT message is intercepted, call your routines to BitBlt a BMP or draw other graphics in the client space. When WM_ERASEBKGND message is intercepted, prevent it from being passed on to VB (all subclassing controls use slightly different terminology for this). If you're using MsgHook, put the statements below in the MDI form's Form_Load event. To see this and other MDI techniques in action, download MDIDMO.ZIP from either the MSBASIC or VBPJ forums on CompuServe and try out the demonstrations.

```
MsgHook1.HwndHook = (GetWindow(Me.hWnd, GW_CHILD))
MsgHook1.Message(WM_PAINT) = True
MsgHook1.Message(WM_ERASEBKGND) = True
```

**—Karl E. Peterson**

## SHOW FREE RESOURCES

It is sometimes useful to determine the amount of Windows resources available. You can show percentage of free resources in an about box, for example, or to detect memory leaks in a program. The latter task requires that you continually update the free-resource display so that you can see when the amount of free resources changes.

The percentage of free system resources is determined by using the Windows API function GetFreeSystemResources. The declarations for this function are:

```
Declare Function GetFreeSystemResources Lib "User"
(ByVal _
   fuSysResource As Integer) As Integer

Global Const GFSR_SYSTEMRESOURCES = &H0
Global Const GFSR_GDIRESOURCES = &H1
Global Const GFSR_USERRESOURCES = &H2
```

To continually update the display, place three labels on your form. Name the labels lblResources and set the indexes to 0, 1, and 2. Next, place a timer on your form called Timer1 and set the interval property to between 500 and 1500. Place this code in the timer event handler:

```
Sub Timer1_Timer ()
   Dim s As String
   s = CStr(GetFreeSystemResources_
      (GFSR_SYSTEMRESOURCES))
   lblResources(0) = "Free System _
      Resources: " & s & "%"
   s = CStr(GetFreeSystemResources_
      (GFSR_GDIRESOURCES))
   lblResources(1) = "Free GDI _
      Resources: " & s & "%"
   s = CStr(GetFreeSystemResources(GFSR_USERRESOURCES))
   lblResources(2) = "Free User _
      Resources: " & s & "%"
End Sub
```

**—Jonathan Wood**

## 3-D TEXT BOXES IN VB PRO

To create a 3-D text box, place a text box in a 3-D Panel and set the panel's AutoSize property to "Autosize Child to Panel." Then set the BevelOuter and BevelWidth properties of the 3-D Panel as desired.

**—Sheridan Software Tech Support**

## USE VB'S TIMER FUNCTION TO OPTIMIZE CODE

I often see questions in the *VBPJ* forum about comparative speeds for different methods of performing actions in code. In many cases, a simple generalization of which is fastest cannot be made. VB's Timer function, however, allows testing your code as you go. Declare a Single variable, TStart! and use this line at the beginning of the code segment you wish to test:

```
TStart! = Timer
```

At the end of the segment, print the elapsed time to the debug window:

```
Debug.print Timer - TStart!
```

By doing such time-testing you will be better able to choose between similar ways of doing things. In most cases, for example, the CStr and Str functions are much faster than the Format or Format$ functions. Likewise, the If/Elseif sequence will execute much faster than similar code using the IIf or Choose functions, probably because of the added type checking done by the functions. But IIf and Choose often allow much shorter, clearer subroutines. By performing simple time tests, you can better decide on the appropriate code style.

**—William Storage**

## MAKE CONTROLS APPEAR 3-D

The latest trend in Windows programs is the use of a 3-D look that makes windows and controls appear to be three-dimensional. As a result, a number of tools for adding the 3-D look to VB applications are available. One of the most popular is THREED.VBX, which ships with the professional edition of VB 3.0.

Adding more VBXs or DLLs to your program just to enhance your program's look, however, may seem like overkill. The Apply3D routine shows how to make a control such as a text box appear sunken or recessed. Apply3D works by painting dark gray lines along the left and top sides of the control, and light gray lines along the bottom and right sides. This gives the control the appearance of being recessed. Note that the routine does not paint inside of the control.

```
Sub Apply3D (myForm As Form, myCtl As Control)
   'Make specified control "sunken"
   'This routine assumes that the mapping mode
   'for myForm is 3-pixel

   myForm.CurrentX = myCtl.Left - 1
   myForm.CurrentY = myCtl.Top + myCtl.Height
   myForm.Line -Step(0, -(myCtl.Height + 1)), _
```

```
      RGB(92, 92, 92)
   myForm.Line -Step(myCtl.Width + 1, 0), _
      RGB(92, 92, 92)
   myForm.Line -Step(0, myCtl.Height + 1), _
      RGB(255, 255, 255)
   myForm.Line -Step(-(myCtl.Width + 1), 0), _
      RGB(255, 255, 255)
End Sub
```

You'll need to set the form's BackColor to gray (&H00C0C0C0&) and set the ScaleMode property to 3—Pixel. To use the Apply3D routine, call it from the form's Paint event. For example, this code shows what the Paint event would look like if you had two text boxes (Text1 and Text2) that you want to show in 3-D:

```
Sub Form_Paint ()
   Call Apply3D(Me, Text1)
   Call Apply3D(Me, Text2)
End Sub
```

**—Jonathan Wood**

## REPLICATE CONTROLS WITHOUT RESELECTING

To replicate a control without having to reselect it each time, press and hold the Ctrl key when selecting a control from the toolbox. VB will not create a control array in this case, but will name each control sequentially, as in Text1, Text2, and so on.

**—Mark Streger**

## SAVING THE POSITION OF A DESIGNER WIDGETS' DOCKABLE TOOLBAR

The best way to save information about an application's options, such as toolbar positioning, is to write the settings to an INI file. To do so, two Windows API functions must be used: GetPrivateProfileInt and WritePrivateProfileString. Here are two functions that load and save settings to and from an INI file:

```
'These two Declare statements go in the
'Declarations section of a module
Declare Function GetPrivateProfileInt Lib "Kernel" _
      (ByVal lpApplicationName As String, _
         ByVal lpKeyName As String, ByVal nDefault _
   As Integer, ByVal lpFileName As String) As Integer
Declare Function WritePrivateProfileString _
      Lib "Kernel" (ByVal lpApplicationName As String, _
         ByVal lpKeyName As String, ByVal lpString _
            As String, ByVal lplFileName As String) _
   As Integer

Sub LoadToolbarInfo (Tb As Control, _
   Section As String, ININame As String)

   Tb.DockRank = GetPrivateProfileInt(Section, _
      "DockRank", Tb.DockRank, ININame)
   Tb.DockRankSeq = GetPrivateProfileInt(Section, _
      "DockRankSeq", Tb.DockRankSeq, ININame)
   Tb.FloatingLeft = GetPrivateProfileInt(Section, _
      "FloatingLeft", Tb.FloatingLeft, ININame)
   Tb.FloatingTop = GetPrivateProfileInt(Section, _
      "FloatingTop", Tb.FloatingTop, ININame)
   Tb.FloatingWidthInBtns = _
      GetPrivateProfileInt(Section, _
      "FloatingWidthInBtns",Tb.FloatingWidthInBtns, _
      ININame)
   Tb.DockStatus = GetPrivateProfileInt(Section, _
      "DockStatus", Tb.DockStatus, ININame)

End Sub

Sub SaveToolbarInfo (Tb As Control, _
   Section As String, ININame As String)

   Dim rc%

   rc = WritePrivateProfileString(Section, _
      "DockStatus", CStr(Tb.DockStatus), ININame)
   rc = WritePrivateProfileString(Section, _
      "DockRank", CStr(Tb.DockRank), ININame)
   rc = WritePrivateProfileString(Section, _
      "DockRankSeq", CStr(Tb.DockRankSeq), ININame)
   rc = WritePrivateProfileString(Section, _
      "FloatingLeft", CStr(Tb.FloatingLeft), ININame)
   rc = WritePrivateProfileString(Section, _
      "FloatingTop", CStr(Tb.FloatingTop), ININame)
   rc = WritePrivateProfileString(Section, _
      "FloatingWidthInBtns", _
      CStr(Tb.FloatingWidthInBtns), ININame)

End Sub
```

This code can be found in the SAMPLES\TBARSAVE directory that's included with Designer Widgets.

**—Sheridan Software Tech Support**

## AUTOMATICALLY DISPLAY THE CONTENTS OF A COMBO BOX

If you've ever wanted to make a combo box automatically display its contents, you know it is next to impossible with VB's SendKey command. The correct way to do this is with a SendMessage API call:

```
Global Const WM_USER = 1024
Global Const CB_SHOWDROPDOWN = (WM_USER + 15)

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


Dim nRet as Long

nRet = SendMessage(combo1.hWnd, CB_SHOWDROPDOWN, _
   1, ByVal 0&)
```

Sending this message will cause the target combo box to display its list of entries just as if you had clicked the down-arrow button on the combo box control.

**—Deepak Agrawal**

## MAKING THE ENTER KEY WORK LIKE A TAB

To allow the Enter key to function as a Tab, enter this code in the KeyPress event of your text boxes:

```
Sub Text1_KeyPress (KeyAscii As Integer)
   If KeyAscii = 13 Then 'Enter
      SendKeys "{Tab}"
      KeyAscii = 0
   End If
End Sub
```

This allows the user to press Enter to move from one text box to the next. This technique will not work if you have a command button on the form whose default property is True. The button will respond to the Enter key before the text box sees it.

**—Karl E. Peterson**

## DATA UPDATING SPEED

When updating the values of several records in a loop, put a BeginTrans, CommitTrans around the loop. This will significantly speed up the update process:

```
BeginTrans
Do Until dsData.EOF
   dsData.Edit
   dsData!sState_cd = "CA"
   dsData.MoveNext
Loop
CommitTrans
```

**—Paul D. Sherriff/*Visual Basic Power Guides***

## CHECKING FOR A PREVIOUS INSTANCE

When working in Windows, it is very easy to lose a minimized icon on the desktop. Or you may forget you have an application running and try to open it again from the Program Manager. There are times, however, when you don't want to run two separate instances of an application. To prevent this you can use the built-in system object App to determine whether another instance of an application is running. Here's a routine you can use in your Form_Load( ) or Main( ) procedures to do this:

```
Sub Form_Load ()
   Dim sCaption As String

   If App.PrevInstance Then
      sCaption = Me.Caption
      MsgBox "Another Instance Is Already Running"
      Me.Caption = "Different Caption"

      AppActivate sCaption
      SendKeys "% R", True

      Unload Me
   End If
End Sub
```

Checking the App.PrevInstance property to see if it contains a True value tells you if another instance is running. If it is, inform the user, then activate the first instance prior to shutting down the second. You can activate any application that is currently running by using the AppActivate statement and passing the text that appears in the application's title bar.

This technique requires that the current instance has the same caption as the one you want to activate, so you'll need to change the current title of the main window and then call AppActivate with the caption of the application. Call the state-



HOME

ment SendKeys to tell the other instance to restore itself to a normal window state. The "% R" string passed to the SendKeys statement tells the application to invoke the control menu and select the Restore option.

The code to check for multiple instances should be placed in the Main( ) or Form_Load( ) of your start-up form. This technique will not work if you incorporate information within the caption at run time, such as the current data-file name.

**—Paul D. Sherriff/*Visual Basic Power Guides***

## USE READ-ONLY ATTRIBUTES WHEN SHARING CODE

If you set a VB file's DOS attribute to read-only, VB will show it with a red lock in the project window and prevent it from being overwritten. This is a handy way of sharing a common code library between developers' projects and ensuring that no one can modify the "common" code.

**—Craig Goren**

## GENERATING A "REAL" TAB IN A TEXT BOX

Here's the code I used to trap the Tab key and use it to generate a "real" tab in a text box:

```
' Trap the tab key to allow tabs in a text box
' This function should be called by the lostFocus
' event for the control that needs to snag the tab
' Parameters:
'  txtControl     text box control

' Setting keyboard info
Declare Function GetKeyState% Lib "User" _
   (ByVal nVirtKey%)

' Virtual key values
Global Const VK_TAB = &H9

Sub snagTab (txtControl As Control)
   Dim retVal As Integer, currSelStart As Long
   retVal = GetKeyState(VK_TAB)

   If retVal = -128 Or retVal = -127 Then
' tab key pressed
      currSelStart = txtControl.SelStart
      If currSelStart = 0 Then
         txtControl.Text = Chr$(9) & txtControl.Text
      Else
         txtControl.Text = Left(txtControl.Text, _
            currSelStart) & Chr$(9) & _
            Mid(txtControl.Text, currSelStart + 1)
      End If
      ' Change the focus back to this control and
      ' reset the current insert point to past
      ' the new "tab"
      txtControl.SetFocus
      txtControl.SelStart = currSelStart + 1
   End If

End Sub
```

**—Deborah Kurata**

## CLEAR OUT DATA WHEN UNLOADING FORMS

When unloading a form, use the following piece of code behind a Close button:

```
Unload <formName>
```

This code should also be put in the Form_Unload( ) event handler for that form:

```
Set <formName> = Nothing
```

This clears out any remaining code/data in the data segment for the form, if you don't do it explicitly. This is important when you load/unload many forms, especially those with lots of controls.

By the way, this is not a bug—it's done this way by design. The same is true for VBA—if you don't initialize global (module) variables in VBA (for Excel, for example) and have this kind of expression:

```
i% = i% + 1
```

then i% will successively have the values 1, 2, 3, . . . as you run the VBA script one more time. Again, by design.

**—Patrick O'Brien**

## HIDE AND SHOW THE SCROLL BAR IN A MULTILINE TEXT BOX

Have you ever wanted to make that pesky scroll bar disappear when it is not needed, and reappear when the text exceeds the viewable area of the text box? Setting the scroll bar properties at run time won't do it. You could use API calls, but how do you decide when the lines have exceeded the viewing area? Here's a set of declarations and procedures that will do just that for you:

```
'=USER TYPES =
Type pointapi
   x As Integer
   y As Integer
End Type
Type RECT   '8 Bytes
   Left As Integer
   Top As Integer
   right As Integer
   bottom As Integer
End Type
Type TEXTMETRIC   '31 Bytes
   TMHeight As Integer
   tmAscent As Integer
   tmDescent As Integer
   tmInternalLeading As Integer
   tmExternalLeading As Integer
   tmAveCharWidth As Integer
   tmMaxCharWidth As Integer
   tmWeight As Integer
   tmItalic As String * 1
   tmUnderlined As String * 1
   tmStruckOut As String * 1
   tmFirstChar As String * 1
   tmLastChar As String * 1
   tmDefaultChar As String * 1
   tmBreakChar As String * 1
   tmPitchAndFamily As String * 1
   tmCharSet As String * 1
   tmOverhang As Integer
   tmDigitizedAspectX As Integer
   tmDigitizedAspectY As Integer
End Type
Global apnt As pointapi
Declare Function SendMessage& Lib "User" _
   (ByVal HWND%, ByVal wmsg%, ByVal wparam%, _
      lparam As Any)
Declare Function GetTextMetrics% Lib "GDI" _
   (ByVal hDC%, lpMetrics As TEXTMETRIC)
Declare Function GetDC% Lib "User" (ByVal HWND%)
Declare Function SelectObject% Lib "GDI" _
   (ByVal hDC%, ByVal hObject%)
Declare Function ReleaseDC% Lib "User" _
   (ByVal HWND%, ByVal hDC%)
Declare Function WindowFromPoint% Lib "User" _
   (ByVal x As Any)
Declare Sub SHOWSCROLLBAR Lib "User" _
   (ByVal HWND%, ByVal wbar%, ByVal bshow%)
'==============API CONSTANTS====================
Global Const WM_USER = &H400
Global Const EM_SETREADONLY = (WM_USER + 31)
Global Const EM_GETLINECOUNT = (WM_USER + 10)
Global Const EM_GETFIRSTVISIBLELINE = _
   (WM_USER + 30)
Global Const EM_GETRECT = (WM_USER + 2)
Global Const WM_GETFONT = &H31
```

This sub is called from your form to make the scroll bars visible or invisible, depending on whether the lines of text have exceeded the viewable area in the text box. You must pass it the hWnd of the text box and the scroll bar type that the text box is using:

```
Sub Scroll_show (thwnd%, bartype%)
'=============================================
'This Subroutine controls the visibility of the
' scrollbars in the designated Edit Control. The
' requirements for this subroutine are:
' BarType%.......Integer selecting scrollbar type
' thwnd%.........Hwnd of the edit control
' API CONSTANTS
'        EM_GETLINECOUNT
'        EM_GETFIRSTVISIBLELINE
' !! MUST BE A TRUETYPE FONT IN USE !!
'=============================================
LinECount% = SendMessage(thwnd%, _
   EM_GETLINECOUNT, 0, 0)
FirstVisible% = SendMessage(thwnd%, _
   EM_GETFIRSTVISIBLELINE, 0, 0)

If LinECount% > GETVISIBLELINES(thwnd%) Then
   SHOWSCROLLBAR thwnd%, 1, True
Else
   SHOWSCROLLBAR thwnd%, 1, False
End If

End Sub
```

This function returns the number of visible lines in a text box when you pass it the hWnd property of the text box:

```
Function GETVISIBLELINES% (thwnd%)
'=========================================
'This function returns the number of visible
' lines in an edit control.  It requires:
' RC.........User Defined Type RECT
' TM.........User Defined Type TEXTMETRIC
' THWND%.....The hwnd of the edit control
' API FUNCTIONS:
'           SendMessage()
'           GetDC()
'           SelectObject()
'           ReleaseDC()
```



HOME

```
' API CONSTANTS:
'           EM_GETRECT
'           WM_GETFONT
'=============================================
Dim RC As RECT
Dim hDC%
Dim LFONT%, OLFFONT%
Dim TM As TEXTMETRIC
Dim DI%

'GET THE RECTANGLE
LC% = SendMessage(thwnd%, EM_GETRECT, 0, RC)
'GET THE FONT HANDLE
LFONT% = SendMessage(thwnd%, WM_GETFONT, 0, 0&)
'GET DEVICE CONTEXT
hDC% = GetDC(thwnd%)
'SELECT LOGICAL FONT
If LFONT% <> 0 Then OLDFONT% = SelectObject_
   (hDC%, LFONT%)
DI% = GetTextMetrics(hDC%, TM)
If LFONT% <> 0 Then LFONT% = SelectObject_
   (hDC%, LFONT%)
'GET VISIBLE LINES
GETVISIBLELINES% = _
   (RC.bottom - RC.Top) / TM.TMHeight
DI% = ReleaseDC(thwnd%, hDC%)
End Function
```

This sub is the one that actually makes the scroll bars visible or invisible, depending on the value of the flag variable. It requires the hWnd property of the control:

```
Sub ListScrollShow (thwnd%, flag As Integer)
'=============================================
'Makes the control's scrollbar visible or
'invisible depending on flag value.
'Flag = True to show
'FlAG = fALSE TO HIDE
'=============================================
   SHOWSCROLLBAR thwnd%, 1, flag
End Sub
```

**—Ibrahim Malluf**

## *IN-LINE CODE*

Calling a function or procedure incurs some overhead. If you have a two-line function that is called from only three places, you might consider putting those three lines in the functions themselves and not make them a function. This can save some processing time. This could lead to some future maintenance problems, however. Here are two options.

Option 1:

```
For iLoop = 1 To 100
   Call DoSomething(iLoop)
Next
Sub DoSomething(iLoop As Integer)
   Print iLoop + 10
End Sub
```

Option 2:

```
For iLoop = 1 To 100
   Print iLoop + 10
Next
```

The two options are very simple examples, but you can see that if you use the statement Print iLoop + 10 in two or more places, you have a maintenance problem if you ever need to change that line. Option 2, however, will run faster than Option 1.

**—Paul D. Sherriff/*Visual Basic Power Guides***

## *COPY MENU STRUCTURES BETWEEN FORMS*

Here's a simple way of copying similar menu structures between forms: If you save a file as text, its menu section can be pasted into a new form's menu section with a text editor. (*For more about menus, please see Deepak Agrawal's article in the January 1995 issue of* Visual Basic Programmer's Journal*—Ed.*)

**—Craig Goren**

## *SET AUTOREDRAW TO FALSE*

If AutoRedraw is set to True, VB keeps a bitmap of the form in memory that it uses to redraw the form after another window in front of it is closed. Because this consumes memory, set AutoRedraw to False if you are not using any graphics methods. Most business applications can have AutoRedraw set to False. This property applies to picture boxes as well as forms.

**—Paul D. Sherriff/*Visual Basic Power Guides***

## USE SPECIFIC OBJECT TYPES

When passing controls as parameters, make the function that is receiving the parameter accept the specific object type instead of generic types. The exception to this is when multiple types of objects may be passed into a routine. In these cases, you must test for the object type within the routine.

Specific object types are Combo Box, List Box, Text Box, and so on. Generic types are Form, Control, MDIForm, and Object.

**—Paul D. Sherriff/*Visual Basic Power Guides***

## LASSOING CONTROLS AND THEIR COMMON PROPERTIES

To lasso controls and set their common properties together, select multiple controls by holding down the Ctrl key while dragging the mouse pointer around the controls, or select multiple controls by holding the Ctrl key while you click on them individually. Pressing F4 brings up a view of the properties windows that shows only the shared properties. Changing a property such as font or color will affect all selected controls.

**—Mark Streger**

## TOGGLE BOOLEAN VALUES USING NOT

To toggle a variable between True and False, use the Not operator instead of an If statement. Use the Not operator on those variables you have explicitly set with the True or False keyword. Option 1 runs slower than Option 2.

Option 1:

```
If bPerform Then
   bPerform = False
Else
   bPerform = True
End If
```

Option 2:

```
bPerform = Not bPerform
```

Be careful when using the Not operator on integers that are a number other than True or False (-1 and 0 respectively):

```
Sub cmdBool_Click ()
   Dim iBool As Integer
   Dim iTemp As Integer

   iBool = True
   Print iBool                ' Prints -1
   Print Not iBool            ' Prints 0

   iTemp = 5
   Print iTemp                ' Prints 5
   Print Not iTemp            ' Prints -6
   If iTemp Then
      Print "iTemp is True"   ' Prints Here
   Else
      Print "iTemp is False"
   End If
End Sub
```

The Not operator does bitwise manipulation, so manipulating the 5 actually makes it a negative 6.

**—Paul D. Sherriff/*Visual Basic Power Guides***

## EDITING GRID CELLS

Grids provide a nice way to present certain types of spreadsheet information. Unfortunately, the grid in VB does not allow editing. With just a little bit of code you can simulate editing on top of a grid.

To begin, you need to create a form with a grid and add a hidden text box. Name the grid control grdFields, and the text box txtEdit.

When the user double-clicks on a cell you need to move the hidden text box over that cell and make it visible. Use this code in the grid's DblClick event:

```
Sub grdFields_DblClick ()
   miLastRow = grdFields.Row
   miLastCol = grdFields.Col
   Call GridEdit(grdFields, txtEdit)
End Sub
```

You'll need two module-level variables to track which row and column the cursor is on. These variables will be used later to replace the information from the text box back into the grid.

To find the location on the screen for the text box, you need to calculate the cell's distance from the top and left of the form. You also need to add each row height and column width individually, because each cell can be a different size. Add this GridEdit() routine to calculate the position and move the text box on top of the grid:

```
Sub GridEdit (grdCurrent As Control, _
      ctlEdit As Control)
```



HOME

```
   Dim iWidth As Single      ' Total Height
   Dim iHeight As Single     ' Total Width
   Dim iLoop As Integer

   iWidth = _
      grdCurrent.Left + Screen.TwipsPerPixelX
   iHeight = _
      grdCurrent.Top + Screen.TwipsPerPixelY

   ' Get Total Width
   For iLoop = 0 To grdCurrent.Col - 1
      iWidth = iWidth + grdCurrent.ColWidth(iLoop)
      If grdCurrent.GridLines Then
         iWidth = iWidth + (Screen.TwipsPerPixelX * _
            grdCurrent.GridLineWidth)
      End If
   Next

   ' Get Total Height
   For iLoop = 0 To grdCurrent.Row - 1
      iHeight = iHeight + grdCurrent.RowHeight(iLoop)
      If grdCurrent.GridLines Then
         iHeight = iHeight + (Screen.TwipsPerPixelY * _
            grdCurrent.GridLineWidth)
      End If
   Next

   ' Move the Text Box On Top Of The Grid
   ctlEdit.Move iWidth, iHeight
   ctlEdit.Height = _
      grdCurrent.RowHeight(grdCurrent.Row)
   ctlEdit.Text = grdCurrent.Text
   ctlEdit.Width = _
      grdCurrent.ColWidth(grdCurrent.Col)
   ctlEdit.Visible = True
   ctlEdit.SetFocus
   ctlEdit.ZOrder 0
End Sub
```

After you've calculated the width and height, use the Move method to place the text box at the proper location. Next, make the text box visible and set its ZOrder to 0, to put it on top of the grid. You also need to move the text from the grid into the Text Box.

After the user finishes editing the text in the text box, you need to put the text back into the grid and hide the text box. Use this in the grid's Click event:

```
Sub grdFields_Click ()
   If txtEdit.Visible Then
      Call GridReset(grdFields, txtEdit, _
         miLastRow, miLastCol)
   End If
End Sub
```

Check to see if the text box is visible. If it is, you need to take the edited data and put it back into the grid with the GridReset( ) procedure:

```
Sub GridReset (grdFields As Grid, _
      ctlEdit As Control, iRow As Integer, _
      iCol As Integer)
   Dim iOldRow As Integer
   Dim iOldCol As Integer

   iOldRow = grdFields.Row
   iOldCol = grdFields.Col

   grdFields.Row = iRow
   grdFields.Col = iCol
   grdFields.Text = ctlEdit.Text
   ctlEdit.Visible = False
   ctlEdit.Move 0, 0

   grdFields.Row = iOldRow
   grdFields.Col = iOldCol
End Sub
```

When the user clicks on another cell after editing the text box, the grid has been updated to the new row and column. Go back to the last row and column where the user was editing, take the contents of the text box, and put that value into the grid's cell. You can then make the text box invisible and move it to an area on the form that is out of the way.

This tip won't work if the grid has scrollbars, except when scrolled all the way to the left and top. To fix this would require additional loops to add up the height/width of fixed rows/columns. In that case, the other loops would need to be adjusted to calculate only from TopRow to Row, rather than from 0 to Row – 1 (but I'd want to test it to be sure!).

Also, it goes nuts if the scrollbar is clicked while text box is visible. At any rate, it will work if scrollbars aren't enabled.

**—Paul D. Sherriff/*Visual Basic Power Guide***

---

## *VB Assist Quick Tips*

Using the AutoSave feature in the VBAssist Options box automatically saves changes made to your project after a defined time interval.

To temporarily bypass VBAssist's Control Lock feature, hold down the Ctrl key while resizing or moving controls.

To prevent VBAssist from automatically loading the last project, hold down the Shift key while double-clicking on the VBAssist program icon in Program Manager.

**—Sheridan Software Tech Support**

## SAVING CHANGES TO A DATAGRID ROW BEFORE THE FORM CLOSES

There are two ways to save changes before exiting a form. One is to invoke the Update method on the data control the DataGrid is bound to from within the QueryUnload event of the form. Another way is to force the update in the Validate event of the data control:

```
Sub Data1_Validate (Action As Integer, _
      Save As Integer)
   If Action = 11 then Save = True
   'if pending changes then save changes
End Sub
```

**—Sheridan Software Tech Support**

---

## SHORTEN CONTROL AND FORM NAMES

Keep control and form names as short as possible. Such names are kept in the EXE file. Variable names, however, are not kept in the EXE.

**—Paul D. Sherriff/*Visual Basic Power Guide***

---

## EASY-TO-IMPLEMENT STATUS BAR WITH SPYWORKS

The SBCEasy custom control that comes with Desaware's SpyWorks-VB can be used to update a status bar whenever the mouse moves over any control or form in your application. All you need to do is set the MouseTransit property to start tracking the mouse with SBCEasy. SBCEasy receives a MouseEnter and MouseExit event each time the mouse enters and exits a control or form. You can add code in these events to update your status bar with the appropriate help text.

The Tag property of a control is one possible place to store the status bar help text. During a MouseEnter event, you can retrieve the string from the Tag property of the control the mouse is currently in and update the status bar. SBCEasy's TransitHctl property contains the VB control handle of the control the mouse is currently in. You can then pass the TransitHctl to the dwGetPropertyValue function to retrieve the Tag property. Here's an example:

```
SBCEasy1_MouseEnter(...)
   dim iresptr%, tagstr$
   :
   tagstr$ = dwGetPropertyValue(SBCEasy1.TransitHctl, _
      "Tag", iresptr%)
   If iresptr% = 0 Then StatusBar.Caption = tagstr$
   :
   :
End Sub
```

**—Daniel Appleman**

---

## USE THE CONTROLS COLLECTION

If you need to reference every control on a form, using the Controls collection will be faster than referencing every control directly. For example, if you have four command buttons on a form and you want to set the Enabled property to False for all of them, you have two options.

Option 1:

```
cmdAdd.Enabled = False
cmdEdit.Enabled = False
cmdDelete.Enabled = False
cmdNext.Enabled = False
```

Option 2 will execute approximately 10 to 15 percent faster than Option 1:

```
For iLoop = 0 To 3
   Controls(iLoop).Enabled = False
Next
```

**—Paul D. Sherriff/*Visual Basic Power Guides***

---

## PREVENTING "INVALID USE OF NULL" ERRORS

If your app uses database or other functions that can return Null variants, your app will crash sooner or later with that dreaded "Invalid Use of Null" error. Preventing it is simple: assign an *empty* string and a variant to your target variable or control:

```
Text1.Text = "" & SomeDynaset("SomeField")
```

**—Michiel de Bruijn**

## ALWAYS USE THESE

Always use Option Explicit: it draws immediate attention to variable typos.

Always use Save As Text: files are less easily corrupted, more easily recovered, and editable by a text editor other than the VB text editor, which is useful for copying code from an old project without shutting down your current VB project. (*Check out Deborah Kurata's article "The Top 10 Do's of Programming," in the December 1994 issue of* Visual Basic Programmer's Journal *for more information on these two tips—Ed.*).

**—Craig Goren**

## TROUBLESHOOTING ERRATIC VBX BEHAVIOR

If a VBX is acting erratically, make sure you are using the most current version of the VBX on your system (and make sure to put it into your user's Windows/System directory too). Use WPS.EXE (Windows Process Status, which is distributed with VB/Pro and placed in the CDK directory) to determine the path that the VBX file is being read from. If it is from a path other than the one you anticipated, chances are that you are using a prior or corrupt version of the VBX.

**—MicroHelp Tech Support**

## MOVE CONTROLS INTO A FRAME

To move controls into a frame, select one or more controls and cut them out using the cut menu option. Select the frame you wish to place the controls into and then paste them in. This technique also can be used when moving controls onto other controls such as Tabs.

**—Mark Streger**

## ELIMINATE DEAD CODE

Be sure to remove functions or procedures that you are no longer using. If you delete a control, be sure to remove the event procedures that were tied to that control.

**—Paul D. Sherriff/*Visual Basic Power Guides***

## SETTING TAB STOPS IN LIST BOXES

To quickly set tab stops in list boxes, use the SendMessage API call. The units used are called Dialog Base Units, which average out to about four per character. Some experimentation is required to get the settings just right, but from then on it's easy. Here's an example subroutine that sets three tab stops in a standard list box:

```
Declare Function SendMessage Lib "User" _
   (ByVal hWnd As Integer, ByVal wMsg As Integer, _
      ByVal wParam As Integer, lParam As Any) As Long
Global Const WM_USER = &H400
Global Const LB_SETTABSTOPS = (WM_USER + 19)
Sub SetTabs (Lst As ListBox)
   ReDim Tabs(0 To 2) As Integer
   Dim Rtn&
   Tabs(0) = 70
   Tabs(1) = 120
   Tabs(2) = 160
   Rtn = SendMessage(Lst.hWnd, LB_SETTABSTOPS, _
      3, Tabs(0))
End Sub
```

Use the API's GetDialogBaseUnits function to calculate the number of units required for a given string. The function's return value contains a number that is four times the average character width of the system font. Thus, to find the width in dialog base units of a string (A$), use the following code:

```
Declare Function GetDialogBaseUnits& Lib "User" ()
DBU& = GetDialogBaseUnits&()
'Extract low word (character width)
W% = DBU& And &HFFFF&
'Calculate width of A$ in dialog base units
DBUWidth% = (Len(A$) * W%) \ 4
```

**—Karl E. Peterson**

## USE THE IMAGE CONTROL INSTEAD OF THE PICTURE CONTROL

Because the picture control has more overhead than the image control, it's best to use the image control when you need to display a graphic. Use the picture control when you need to contain other controls, align the picture either to the top or bottom, or use graphics methods. The image control optimizes both for speed and size and consumes none of the GDI heap (one of the most critical of the so-called "system resources").

**—Paul D. Sherriff/*Visual Basic Power Guides***

## Retrieving Data from a DataGrid Row that Isn't Current

When you change the row in the data control that a Sheridan DataGrid is bound to, the DataGrid will also reposition its current row. To avoid this, use the Clone method to make a copy of the dynaset in the data control. Then, accessing the rows in the cloned dynaset will not affect the current row in the DataGrid. Here's an example:

```
Dim DynClone As Dynaset
Set DynClone = Data1.Recordset.Clone()
'clone the data control's dynaset
DynClone.MoveFirst
Do While Not DynClone.EOF()
   ...perform operation on the current _
      row of DynClone...
   DynClone.MoveNext
Loop
```

**—Sheridan Software Tech Support**

## Menu Status on MDI Forms with SpyWorks-VB

Many features of SBCEasy (a tool that comes with Desaware's SpyWorks-VB) work only on its container, and because it cannot be placed on a MDI Parent form, the MenuSelect event does not work on MDI forms. Fortunately, it is quite easy to use SBC.VBX to implement the same functionality.

First, create MDI Parent and MDI Child windows and place a PictureBox control onto the MDI Parent. Place an SBC SubClass control on top of that PictureBox. In the Parent's Load event, set the window to be subclassed:

```
SubClass1.HwndParam = MDIForm1.hWnd
```

Adjust the SubClass control's Message property to intercept WM_MENUSELECT. Then, in the SubClass control's WndMessage event, get the caption of the selected menu item with this code:

```
Sub SubClass1_WndMessage (wnd As Integer, _
      msg As Integer, wp As Integer, lp As Long, _
      retval As Long, nodef As Integer)
   Dim id%, di%
   Dim hmenu%, menuflags%
   Dim menustring$

   dwDWORDto2Integers lp, menuflags%, hmenu%
   id% = wp
   ' Ignore bitmaps, popups and system menu for now
   If menuflags And (MF_BITMAP Or MF_POPUP Or _
         MF_SYSMENU) Then
      Form1.Label1.Caption = ""
      Exit Sub
   End If
   menustring$ = String$(32, 0)
   ' Get the string
   di% = GetMenuString(hmenu%, id%, menustring$, _
      31, MF_BYCOMMAND)
   ' Strip off anything past the null termination
   menustring$ = dwGetStringFromLPSTR$(menustring$)

   ' This menu string can be used as you wish (for
   ' example, to update status bars)
End Sub
```

**—Daniel Appleman**

## Clearing the Contents of a DataGrid

If the DataGrid is bound to a VB data control, you need to set the RecordSource property of the data control to "" (null string) and then use the Refresh method:

```
Sub Command1_Click()
   Data1.RecordSource = ""
   'set RecordSource to null string
   Data1.Refresh
   SSDataGrid1.Refresh
End Sub
```

If the DataGrid is unbound, setting the Rows property to zero will clear it.

**—Sheridan Software Tech Support**

## Set ClipControls Property to False

Setting the ClipControls property to False significantly reduces the time it takes forms to paint. The default is True, so you'll need to change it to False. If you're using graphics methods, however, you may not be able to do so. The ClipControls property is applied to forms, frames, and picture controls.

**—Paul D. Sherriff/*Visual Basic Power Guides***



HOME

## TIPS ON USING BOOLEAN LOGIC

Programmer's of all skill levels often make errors when using Boolean logic. This statement might not evaluate the way you think it should:

```
IF (SomeNumber AND 16) OR _
   (SomeOtherNumber <> 0) THEN...
```

The (SomeNumber AND 16) will never return a True (-1). It will return False (0) or <> False (some value). Always phrase your evaluations in a TRUE, FALSE, or <> ZERO (<> ZERO in this case means "has value" as opposed to NOT FALSE which means TRUE (-1)). Not only will your logical intention be better understood, you'll be less likely to experience a logic fault that would be a bear to track down.

**—MicroHelp Tech Support**

---

## CALL THE CLICK EVENT

If you need to fire a command button's Click event, you can set the Value property to True:

```
cmdAdd.Value = True
```

The real benefit of this is that you can call code in other form modules. It is faster, however, to call the event procedure directly:

```
Call cmdAdd_Click
```

This is true for all controls, not just the command button.

**—Paul D. Sherriff/*Visual Basic Power Guides***

---

## PLACE A HORIZONTAL SCROLLBAR ON A LIST BOX

I hate it when my standard VB list box or combo box has entries that extend too far to the right and seem truncated. Did you know that you can display a horizontal scrollbar on the list box so that you can scroll to the right? Just issue the SendMessage API function like this:

```
Global Const WM_USER = 1024
Global Const LB_SETHORIZONTALEXTENT = _
   (WM_USER + 21)

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

Dim nRet As Long
Dim nNewWidth as Integer

nNewWidth = list1.width + 100
   'New width in pixels
nRet = SendMessage(list1.hWnd, _
   LB_SETHORIZONTALEXTENT, nNewWidth, ByVal 0&)
```

**—Deepak Agrawal**

---

## GET RID OF UNUSED DECLARE STATEMENTS

When you use the Declare statement, approximately 11 bytes are added to the size of your EXE. The name of the function and library where it resides are also stored in the EXE.

**—Paul D. Sherriff/*Visual Basic Power Guides***

---

## DISABLING ALL CONTROLS ON A FORM

If you ever need to disable all the controls on a form you can loop through the control array and set each Enabled property to False with this code:

```
Sub cmdArray_Click ()
   Dim iLoop As Integer

   For iLoop = 0 To Me.Controls.Count - 1
      Me.Controls(iLoop).Enabled = False
   Next iLoop
End Sub
```

An alternatively is to set the Enabled property of the form to False, which effectively disables the entire form:

```
Me.Enabled = False
```

There is a side effect to the second method, however: because you can't use the control menu from the form, there is no way to close the form. You'll need to have another form unload this disabled form.

**—Paul D. Sherriff/*Visual Basic Power Guides***

## GIVING FORMS A 3-D LOOK

It's not widely known that you can use CTL3D.DLL (or CTL3DV2.DLL) to give your VB forms the same 3-D look of Microsoft's Office suite of applications.

First, your app must register with CTL3D at startup and deregister before it ends. The Ctl3DInit and Ctl3DExit functions take care of this when you pass them the hWnd property of any form in your program. To add the 3-D effect to a form, call the Ctl3DForm subroutine with the form's name as the parameter.

Ctl3DForm works its magic by calling GetWindowLong and SetWindowLong to set the form's DS_MODALFRAME style bit. Then it calls Ctl3DSubclassDlgEx to connect the form to CTL3D's drawing routines:

```
DefInt A-Z
Option Explicit
Global Const BUTTON_FACE = &H8000000F
Global Const FIXED_DOUBLE = 3
Global Const DS_MODALFRAME = &H80&
Global Const GWL_STYLE = (-16)
Global Const GWW_HINSTANCE = (-6)
Declare Function Ctl3dAutoSubclass Lib _
   "CTL3D.DLL" (ByVal hInst)
Declare Function Ctl3dSubclassDlgEx Lib _
   "CTL3D.DLL" (ByVal hWnd, ByVal Flags&)
Declare Function Ctl3dRegister Lib _
   "CTL3D.DLL" (ByVal hInst)
Declare Function Ctl3dUnregister Lib _
   "CTL3D.DLL" (ByVal hInst)
Declare Function GetWindowLong& Lib "User" _
    (ByVal hWnd, ByVal nIndex)
Declare Function GetWindowWord Lib "User" _
    (ByVal hWnd, ByVal nIndex)
Declare Function SetWindowLong& Lib "User" _
    (ByVal hWnd, ByVal nIndex, ByVal dwNewLong&)

Sub Ctl3DInit (hWnd As Integer)

   Dim hInst As Integer
   Dim iResult As Integer

   hInst = GetWindowWord(hWnd, GWW_HINSTANCE)
   iResult = Ctl3dRegister(hInst)
   iResult = Ctl3dAutoSubclass(hInst)

End Sub

Sub Ctl3DExit (hWnd As Integer)

   Dim hInst As Integer
   Dim iResult As Integer

   hInst = GetWindowWord(hWnd, GWW_HINSTANCE)
   iResult = Ctl3dUnregister(hInst)

End Sub

Sub Ctl3DForm (frm As Form)

   Dim hWnd As Integer
   Dim iResult As Integer
   Dim lStyle As Long

   hWnd = frm.hWnd
   If frm.BorderStyle = FIXED_DOUBLE Then
      frm.BackColor = BUTTON_FACE
      lStyle = GetWindowLong(hWnd, GWL_STYLE)
      lStyle = lStyle Or DS_MODALFRAME
      lStyle = SetWindowLong(hWnd, GWL_STYLE, lStyle)
      iResult = Ctl3dSubclassDlgEx(hWnd, 0)
   End If

End Sub
```

Ctl3DForm will only "3-D-ize" forms whose BorderStyle is 3 (FIXED_DOUBLE). You can achieve some, uh, *unusual* effects by trying it on forms with other BorderStyles.

**—Phil Weber**

---

## TROUBLESHOOTING CUSTOM DLLS IN VB

If you're getting GPFs and you aren't sure if the cause is your custom DLL or VB, write a BAS file stub that simulates your DLL. If the GPFs persist, its not your DLL.

**—Craig Goren**

---

## RUN MULTIPLE COPIES OF VB

For debugging or whenever you want to copy code from one project to another, it would be very easy if you could have two instances of VB running at the same time. Unfortunately, Microsoft decided to let you start it only once. Short of getting a second machine or running NT, there's a simple solution: make a copy of your VB.EXE called, for example, VB1.EXE. If you run VB.EXE first, you can start VB1 and have a second copy running. Note that doing things the other way around—starting VB1 and then VB—won't work.

**—Michiel de Bruijn**

## ACCESS KEYS WITH LABELS

There is a very simple trick you can perform with labels and text boxes that will help users who like to use the keyboard instead of the mouse. Notice the underlines in the label controls. By pressing the Alt key in combination with any of the underlined keys, the cursor will jump to the text box next to this label. For example, pressing Alt-F will move to the text box next to First Name, and pressing Alt-C will move the cursor to the City text box.

To make this happen, set a TabIndex property like this:

| Prompt | Label Control | Text Box Control |
|---|---|---|
| First Name | 0 | 1 |
| Cust. Type | 2 | 3 |
| Last Name | 4 | 5 |
| Street | 6 | 7 |

and so on....

Label controls cannot receive the focus. When VB detects that you have pressed an Alt-key combination that corresponds to a label, the focus is set to the next control in the tab order that can take the focus. Making the TabIndex property of the text box one greater than the label control will cause the text box to become active when the key combination for the label is pressed.

**—Paul D. Sherriff/*Visual Basic Power Guides***

## USE THE LEN( ) FUNCTION TO CHECK FOR EMPTY STRINGS

Use the Len( ) function to check for an empty string rather than the "<>" or "=" operators. For instance:

```
If sName <> "" Then
   ...
End If
```

This is functionally the same, but is faster:

```
If Len(sName) Then
   ...
End If
```

**—Paul D. Sherriff/*Visual Basic Power Guides***

## MAKE A READ-ONLY TEXT BOX WITHOUT GRAYING THE TEXT

There may be situations in which you want to display text that the user cannot edit, but a label control doesn't quite fit the bill. What you need is a read-only text box, which is done setting a text box's Enabled property to False. Unfortunately, this also grays the text. An alternative is to place the text box on a picture box control and then set the picture box's Enabled property to False. This technique will also disable the text box's scroll bars if it has any. Another approach is to make the text box read-only by sending a EM_SETREADONLY message to the text box using the API function SendMessage as shown here:

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

Global Const WM_USER = &H400
Global Const EM_SETREADONLY = (WM_USER + 31)

Sub Form_Load ()
   Dim i As Long

   'Prevent user from editing text box
   i = SendMessage(Text1.hWnd, EM_SETREADONLY, _
      True, ByVal 0&)
End Sub
```

This technique allows the user to still select, copy, and scroll the contents of the text box but not edit it.

**—Jonathan Wood and Barry Seymour**

## AN EASY WAY TO QUIT HELP AND RETURN TO VB

When creating help files for VB programs it is useful to let the user easily and quickly leave the help window and return to the VB application. One way to do this is to allow the Escape key to terminate the help program. To do this, add the following code to the [CONFIG] section of the help project file. The code creates an accelerator key (Esc) to invoke the help system macro EXIT, which terminates the help program.

```
[CONFIG]
;lets the ESC key terminate the help program
AddAccelerator(0x1B,0,"Exit()")
```

**—Chuck Peper**

## INDICATING SELECTED/ DESELECTED STATES FOR ITEMS

You can set .SelectedColor and .FillColor in the Mh3DList to be the same, and use .ListPicture and .ListPictureSel properties to indicate selected/deselected states.

**—MicroHelp Tech Support**

## USE SMALLER GRAPHICS TO SAVE RESOURCES

Larger bitmaps used for .ListPicture/ListPictureSel properties in Mh3DList consume proportionally larger amounts of resources. Also, avoid using 256-color bitmaps in controls—they too consume more resources.

**—MicroHelp Tech Support**

## MAKE FORMS STAY ON TOP

To set a form to be always on top, use the subroutine listed here. Pass it the hWnd property of the form you want to float. The OnTop parameter is used to toggle the attribute. If True, floating is turned on; if False, the form will not float.

```
Declare Sub SetWindowPos Lib "User" _
   (ByVal hWnd As Integer, _
   ByVal hWndInsertAfter As Integer, _
   ByVal X As Integer, ByVal Y As Integer, _
   ByVal cx As Integer, ByVal cy As Integer, _
   ByVal wFlags As Integer)
Global Const SWP_NOSIZE = &H1
Global Const SWP_NOMOVE = &H2
Global Const HWND_TOPMOST = -1
Global Const HWND_NOTOPMOST = -2
Sub FormOnTop (hWnd%, OnTop%)
   If OnTop Then
      Call SetWindowPos(hWnd, HWND_TOPMOST, 0, _
      0, 0, 0, SWP_NOSIZE Or SWP_NOMOVE)
   Else
      Call SetWindowPos(hWnd, HWND_NOTOPMOST, 0, _
      0, 0, 0, SWP_NOSIZE Or SWP_NOMOVE)
   End If
End Sub
```

**—Karl E. Peterson**

## FINDING A STRING IN A COMBO BOX

The CBFindString( )procedure searches for a string in a combo box by using the SendMessage( ) API function to find a specific entry in the list. This is much more efficient than searching using VB code:

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

Sub CBFindString (ctlEdit As Control, _
      sSearch As String)
   Dim lPos As Long

   Const CB_FINDSTRING = &H40C
   lPos = SendMessage(ctlEdit.hWnd, CB_FINDSTRING, _
      0, ByVal sSearch)
   If lPos >= 0 Then
      ctlEdit.ListIndex = lPos
   End If
End Sub
```

**—Paul D. Sherriff/*Visual Basic Power Guides***

## USE VSHARE.386

When you're using OLE or the JET database engine, you'll need to load SHARE on your system and all systems you deliver your app to. In the latter case, you are guaranteed to have trouble with users who load SHARE.EXE but do not include the required switches (/F:4096 /L:500)—this can crash your app. To prevent this and many other problems, provide your users with VSHARE.386 instead. This file is freely available from Microsoft on CompuServe and comes with a read-me file that outlines usage and installation.

**—Michiel de Bruijn**

## INDENTING CODE BLOCKS

You can highlight a block of text in the code window and press the Tab key to indent the block or Alt-Tab to unindent it.

**—Craig Goren**



HOME

## BREAK UP LARGE APPS

If you have a large app, consider breaking it down into smaller apps. Use one main EXE that calls the others and DDE to provide communication. When a sub-app opens, open a DDE link to the main app. You can use scripting to enable the apps to communicate with each other to pass data or start processes. Doing this has several advantages including lower memory requirements and lower resource usage. One *big* advantage: Because your app is composed of multiple EXEs, Windows gives each piece its own share of the time slices, so your app runs faster.

**—MicroHelp Tech Support**

## USE BYVAL WHEN API CALLS CAUSE PROBLEMS

If an API call is not achieving the desired or expected effects, try placing ByVal in front of parameters. Likely ones to cause trouble are strings and anything declared "As Any." The APIs that trip people up the most are the various INI file calls, SendMessage, and HMemCpy. Be very suspicious any time a parameter is declared As Any rather than as an explicit type, or if a string isn't declared ByVal.

**—Karl E. Peterson**

## USE A CONTROL'S IMPLICIT VALUE "DEFAULT" PROPERTY

Every control has an implicit "value" property. For text boxes it is the Text property. For Labels it is Caption:

```
lblZip.Caption = "Zip Code"
```

You do not need to reference this property to set the value for the control.

This will execute 10 to 15 percent faster, but you will lose a little readability:

```
lblZip = "Zip Code"
```

**—Paul D. Sherriff/*Visual Basic Power Guides***

## AVOID THE OUTLINE CONTROL

The Outline control might at first seem very usable. Well, it isn't. Apart from the arcane interface, it's almost guaranteed to blow up your app if it's running on a "nonstandard" video driver (such as one with more than 256 colors or one with minor bugs). There are several good third-party Outline replacements that will save you a great number of tech-support calls.

**—Michiel de Bruijn**

## CHANGING COLORS AND FONTS OF DATAGRID CELLS

Attributes such as the foreground color, background color, and fonts of cells in Sheridan's DataGrid can easily be changed by setting RowCellxxxx properties such as RowCellForeColor, RowCellBackColor, RowCellItalic, and so on from within the RowLoaded event. This event fires when the grid initially loads records and while scrolling through rows, allowing you to set various properties for each row in the DataGrid. This code will set column 0's background color to red, text color to white, and font to italics:

```
Sub SSDataGrid1_RowLoaded (BookMark As String, _
   RowNum As Long)
   SSDataGrid1.RowCellForeColor(0) = _
      RGB(255,255,255)
   'set foreground to white
   SSDataGrid1.RowCellBackColor(0) = RGB(255,0,0)
   'set background to red
   SSDataGrid1.RowCellItalics(0) = True
   'set font to italics
End Sub
```

Another way to change the appearance of individual cells in the DataGrid is to set the EvalRowNumber property to a specific row number and then set the appropriate RowCellxxxx properties. This illustrates this method in the Click event of a Command button:

```
Sub Command1_Click()
   SSDataGrid1.EvalRowNumber = 10
   'row to be manipulated
   SSDataGrid1.RowCellForeColor(2) = _
      RGB(255,255,255)
   'set foreground at column 2 to white
   SSDataGrid1.RowCellBackColor(2) = RGB(255,0,0)
   'set background to red
   SSDataGrid1.RowCellItalics(2) = True
   'set font to italics
End Sub
```

**—Sheridan Software Tech Support**

## BUILD YOUR OWN FLOATING TOOLBAR

Ever wanted a floating toolbar? To make one form "owned" by another, all it takes is a simple API call. Afterwards, the owned form will float above the form that owns it, and will be automatically hidden whenever the owner form is minimized. To set up such a relationship, use SetWindowWord with the constant SWW_HPARENT:

```
Declare Sub SetWindowWord Lib "User" (ByVal _
   hWnd%, ByVal nCmd%, ByVal nVal%)
Global Const SWW_HPARENT = -8
Call SetWindowWord(frmOwned.hWnd, SWW_HPARENT, _
   frmOwner.hWnd)
```

**—Karl E. Peterson**

---

## USE THE SHORTEST VARIABLES

Use the shortest data type you can for variables. If you're going to be counting in a loop from 1 to 10, for instance, use an Integer not a Double.

**—Paul D. Sherriff/*Visual Basic Power Guide***

---

## KEYBOARD SHORTCUTS

Yes, these shortcuts are in the manual, somewhere, but it's helpful to brush up on them. Using keystrokes is usually faster than using the mouse to do the same thing. Photocopy these tips and stick them to your monitor until you've memorized them:

**Code Window Control Menu Box**

Alt and Dash accesses the control menu box of a code window. Thus, you can use these keystrokes:

- Alt-Dash-x: Maximize the current code window.
- Alt-Dash-n: Minimize the current code window.
- Alt and F4: Close the code window.

**Navigating through Code Window Text**

- Ctrl and Home: Move to the start of code window text.
- Ctrl and End: Move to the end of code window text. Add the Shift key to select an entire code window: Ctrl+Home, then Ctrl+Shift+End.
- Home: Move to the left end of the current line.
- Shift and End: Select the entire line.
- Ctrl and Right arrow: Move one word to the right (add Shift to select).
- Ctrl and Left arrow: Move one word to the left (add Shift to select).

**Navigating through Procedures**

- Ctrl and Up: Move to the previous procedure in the current code window.
- Ctrl and Down: Move to the next procedure in the current code window.
- F2: Open the View Procedures window for moving directly to other procedures, either in the current module or in other modules.
- Alt and Space: Accesses the control menu box of the main VB menu window (or of any nonchild Windows window).

**—Barry Seymour**

---

## CHANGE THE CONTAINER OF A CONTROL

I use bounded text boxes within the picture control but they act as if they belong to the form. Here's how to make them act as if they belong to a container control (such as a frame):

- Highlight the text boxes by Shift-clicking.
- Cut them into the clipboard.
- Highlight the picture box.
- Paste the text boxes into it.

**—Karl Peterson**

---

## SORTING DATA WITHIN A DATAGRID

Data is sorted by using the "ORDER BY" clause in a SQL statement and then assigning the SQL statement to the RecordSource property of the data control that the DataGrid is bound to. This code displays the Author table in the DataGrid and sorts it by the LastName field:

```
Sub Form1_Load()
   Data1.RecordSource = "Select * from Authors _
      Order By LastName"
   Data1.Refresh
End Sub
```

**—Sheridan Software Tech Support**



HOME

## USING LOSTFOCUS EVENTS WITH COMMAND BUTTON PROPERTIES

If you're using LostFocus events but would also like to set the Default or Cancel properties for command buttons, you'll run into a confusing situation. Those properties trigger the Click event when the user presses either Enter or Escape *without* ever transferring focus to the command button. To trigger the "missing" LostFocus, you need to explicitly transfer focus yourself, and then call DoEvents to resequence to a VB chain of events. (Don't be scared by the naysaying over DoEvents—it's extremely useful in this situation and can cause no ill effects!) Use code similar to this:

```
Sub Command1_Click ()
   Command1.SetFocus
   DoEvents
   If Not (ActiveControl Is Command1) Then
      'Focus was transfered elsewhere
      'by validation code
      Exit Sub
   Else
      'Proceed with Click event code
   End If
End Sub
```

**—Karl E. Peterson**

## KEEP THE CONTROL BAR AND TOOLBOX ON TOP

When a VB form is maximized, you can't normally access the toolbox or VB control bar unless you resize the form to display the VB tools. Here is a simple trick to keep a VB form maximized but have access to all of VB's tools: Press Alt and Escape. VB's control bar and any open windows (toolbox, code modules, properties window, and so on) will appear on top of the maximized form.

**—Deepak Agrawal**

## EDITING A MAK FILE

VB 3.0 does not give you an easy way to edit the contents of your MAK files. This becomes important when you add a new VBX in the middle of a current project, or worse, when you use a different PC that does not have the VBXs your app needs. Such situations, and others, usually trigger errors that hinders the communication channel between your app and Windows.

A solution to this problem is to use a generic text editor, such as Notepad or Write, to "manually" adjust the app's MAK file. For instance, if a VBX file is not in the Windows directory or a project FRM or BAS file is not in your current project directory, correct the file's path as displayed in the MAK file to reflect the location of that file. This technique has saved me lots of headaches in projects with tight deadlines.

**—John D. Conley III**

## USING THE MH3D CALENDAR CONTROL FOR DATE MATH

One of the great unsung heroes in MicroHelp's VBTools is the Mh3dCalendar control. Place a hidden calendar control on one of your main forms. With the .Day, .Month, .Year, and .DateFormat properties, you can do date math and date conversion from anywhere in your application.

**—MicroHelp Tech Support**

## FIND LOST CONTROLS

If a control gets lost on your form, you can select it by choosing its name from the drop-down list at the top of the properties window. Set its Left and Top properties to 0. Choose Bring To Front from the Edit menu. If it's still lost, choose Delete from the Edit menu and create it again.

**—Craig Goren**

## USE THE RIGHT TOOL FOR THE JOB

If you're running into a roadblock in a particular section of your program, step back and take another look at what you're doing. There's a good chance you're using the wrong tool (control, VBX, DLL) for the job. Yes, you can put a screw in with a hammer but it works a lot better with a screwdriver. See if there is a DLL or VBX available that will do what you're wanting—you can save a lot of headaches this way.

**—MicroHelp Tech Support**

## DATA ACCESS SPEED

When you connect to remote servers, always use tables attached to the server through Access. This will significantly speed up your retrieval time. Once a table is attached to a remote server, the whole structure of the table is brought down to your local machine. If it is not attached, the table structure is brought down the line, followed by your data, every time you make a query.

**—Paul D. Sherriff/*Visual Basic Power Guides***

## DEBUGGING WITHOUT DISTURBING WHAT HAS FOCUS

Placing the DEBUG.PRINT expression at strategic places in you program can be a big help in debugging, because code will be written to the immediate window without disturbing what has focus. The DEBUG statement will be ignored when you build an EXE.

**—Mark Streger**

## USE THE VB KNOWLEDGE BASE

Get the Visual Basic Knowledge Base from Microsoft. It contains hundreds of ideas (with code samples). I use it daily to get my job done (and find out new things to try that I never would have dreamed possible). You can get this file from the Microsoft Download Service or from CompuServe (GO MSL).

**—MicroHelp Tech Support**

## QUICKLY EVALUATE AN EXPRESSION OR VARIABLE

Here's how to quickly evaluate an expression or variable: While in debug mode, use the Add Instant Watch dialog to quickly see the current value of an expression in your code. Highlight the variable or expression and press Shift-F9, which opens this dialog. You'll see the expression, plus it's current value. This is much quicker than typing the expression in the debug window.

**—Barry Seymour**

## MAKE A GAUGE WITH MICROHELP'S CONTROLS

You can make a beautiful gauge by combining two MicroHelp controls. Place a Mh3d control onto your form and set the inner bevel just inside of the outer bevel. Place a Mh3dCommand as a child of the Mh3d control with it's top, left , and bottom just inside of the inner bevel of the Mh3d control and its .Width set to 0. Set the .Picture property of the Mh3dCommand to WINLOGO.BMP, set the .WallPaper property to 2 - Replicate, set the .FontStyle property to 3 - Lowered, and set the .Alignment property to 2 - Center. Now add this function to a global module:

```
Sub SetGauge (Ctrl1 As MhThreed, _
   Ctrl2 As Mh3dCommand, percent As Integer)

   MaxWidth = Ctrl1.Width - (Ctrl2.Left * 2)
   Ctrl2.Width = MaxWidth * percent / 100
   Ctrl2.Caption = Str$(percent) + "%"

End Sub
```

You can call this function to set the % fill on the gauge from anywhere. The end effect is a gauge that appears to be engraved in marble. You can use your own bitmaps for other textures such as wood and granite.

**—MicroHelp Tech Support**

## PREVENTING "FILE NOT FOUND"' ERRORS WHEN USING IIF

Although it is undocumented and illogical, using IIF requires you to distribute MSAFINX.DLL with your program.

**—Michiel de Bruijn**

## DRAWING DIRECTLY ON THE FORM SAVES RESOURCES

You can drop labels and frames out of your application (and their accompanying resource and memory usage) by writing a small routine to use Print and Line that draws directly onto your form. Doing so increases speed, saves resources and memory, and uses fewer controls.

**—MicroHelp Tech Support**



HOME

## FIND ENTRIES IN LIST AND COMBO BOXES

Searching for an entry in a list box or a combo box involves scrolling through the list to find a match—a time-consuming process if there are many entries in the list. Here is the old way to search for an entry:

```
Dim nLoop As Integer
'Loop variable, current position of entry in list
Dim sEntry As String
'String being searched
Dim nIndex As Integer
'Location of entry in the list

   sEntry = "J"
   nIndex = -1
   For nLoop = 0 To list1.ListCount - 1
      If sEntry = list1.List(nLoop) Then
         nIndex = nLoop
         Exit For
      End If
   Next
```

A faster way to do this is to send a CB_FINDSTRINGEXACT or LB_FINDSTRINGEXACT with the API function SendMessage. This will return the index of the entry if it is found in the box:

```
Global Const WM_USER = 1024
Global Const CB_FINDSTRINGEXACT = (WM_USER + 24)
'for Combobox only
Global Const LB_FINDSTRINGEXACT = (WM_USER + 35)
'for Listbox only

Declare Function SendMessage Lib "User" _
   (ByVal hWnd As Integer, ByVal wMsg As Integer, _
      ByVal wParam As Integer, lParam As Any) As Long
   Dim sSearch As String
   Dim nIndex As Long

   sSearch = "<Some Text>"
   nIndex = SendMessage(List1.hWnd, _
      LB_FINDSTRINGEXACT, -1, ByVal sSearch)

   If nIndex < 0 Then
      MsgBox "Not Found"
   Else
      'Make matching item the selected one
      List1.ListIndex = nIndex
   End If
```

Windows also has CB_FINDSTRING and LB_FINDSTRING. Try these constants if you want to search for strings that start with the specified string.

**—Deepak Agrawal**

## MAKING MDI CHILDREN INVISIBLE ON LOADING

Contrary to popular belief, an MDI child does not need to be immediately visible at load time. If the Visible property is set to False at *design* time, the child will not become visible until either the last line of its Form_Load event or another statement (such as Show) causes it to do so.

**—Karl E. Peterson**

## VALIDATING TEXT ENTRIES WHEN A DATACOMBO LOSES FOCUS

When setting focus to another control, the DataCombo in Sheridan's Data Widgets can be forced to validate the text portion against its list by referencing the DataCombo's IsItemInList property as follows:

```
If Not DataCombo1.IsItemInList Then
'Check if item is in list
   MsgBox "Invalid Data Entered"
   'If not, display an error
   'then do some additional processing
End If
```

**—Sheridan Software Tech Support**

## SPEED UP LIST BOX UPDATES

When adding a large number of items to a list box, you can greatly speed up the process by disabling redraws of it. A quick call to SendMessage does the trick. Sandwich one to turn off redraws and one to turn them back on around a call to the routine that fills the list box, as shown here. Another method is to set the list box's Visible property to False, but that may not offer as clean an appearance.

```
Declare Function SendMessage Lib "User" _
   (ByVal hWnd As Integer, ByVal wMsg As Integer, _
      ByVal wParam As Integer, lParam As Any) As Long
Global Const WM_SETREDRAW = &HB
nRet& = SendMessage(List1.hWnd, WM_SETREDRAW, _
   False, 0&)
   Call FillMyList(List1)
nRet& = SendMessage(List1.hWnd, WM_SETREDRAW, _
   True, 0&)
```

**—Karl E. Peterson**

## AVOID USING VARIANTS

Variants take more memory and it's slower to get/set their values than other data types. Option 1 will run slower than Option 2.

Option 1:

```
Dim iLoop As Variant
For iLoop = 1 To 100
   Print iLoop
Next
```

Option 2:

```
Dim iLoop As Integer
For iLoop = 1 To 100
   Print iLoop
Next
```

**—Paul D. Sherriff/*Visual Basic Power Guides***

## NEVER HAVE AN UNTRAPPED ERROR IN VB

To prevent untrapped errors in VB, put an error handler in the code for every control/form's events. Unless you want more granularity, you won't need to trap errors in function or module routines.

**—Craig Goren**

## DISPLAY A VB FORM'S SCROLL BARS

A VB form does not automatically display scroll bars, so you may purchased one of the third-party virtual form custom controls. Even without such a tool, you can display a VB form's scroll bars. In the Load event of the target form, call the ShowScrollBar subroutine. You can display the vertical scroll bar, the horizontal scroll bar, or both:

```
Const ZERO = 0
Const NONZERO = ZERO + 1
Const SB_HORZ = 0
Const SB_VERT = 1
Const SB_BOTH = 3
Declare Sub ShowScrollBar Lib "User" _
      (ByVal hWnd As Integer, ByVal wBar As Integer, _
         ByVal bShow As Integer)

   Call ShowScrollBar(Me.hWnd, SB_BOTH, NONZERO)
```

To imitate a scrolling form, you'll need to subclass the form and capture the WM_VSCROLL and WM_HSCROLL messages and set the scroll bar extents via a set of API calls (refer to Chapter 15, page 673 of Daniel Appleman's *Visual Basic Programmer's Guide to the Windows API* (Ziff-Davis Press) for more information).

**—Deepak Agrawal**

## HIDING MDI CHILDREN

MDI children *can* be hidden! Although VB doesn't directly support this, you can use the ShowWindow API call to do so. A simple call like this will do it:

```
Declare Function ShowWindow Lib "User" _
   (ByVal hWnd As Integer, ByVal nCmdShow _
      As Integer) As Integer
Global Const SW_HIDE = 0
Ret% = ShowWindow(frmMDIChild.hWnd, SW_HIDE)
```

Other issues need to be addressed if you use this technique, such as what happens if the active child has a menu when it's hidden or if the hidden child was maximized. These and other pitfalls are covered in a demonstration of MDI techniques, MDIDMO.ZIP, which can be downloaded from either the MSBASIC or *VBPJ* Forums on CompuServe.

**—Karl E. Peterson**

## LOAD VBX/OCX/DLLS IN THE WINDOWS SYSTEM DIRECTORY

It's best to load all shared (or public) VBXs, DLLs, and OCXs in the Windows system directory. Doing so prevents component conflicts between shared components that are loaded in the wrong places. By the way, Windows looks in memory for loaded code modules from VBXs before it looks in the Windows system directory.

**—*VBPJ* Staff**



HOME

# ABOUT THE AUTHORS

**Deepak Agrawal**
is president of DAConsulting Inc., a consulting and training firm specializing in client/server applications and corporate downsizing.
CompuServe: 73322,1561

**Daniel Appleman**
is the author of *Visual Basic Programmer's Guide to the Windows API* (Ziff-Davis Press) and a Contributing Editor of *Visual Basic Programmer's Journal*. He is also president of Desaware (San Jose, Calif.), maker of SpyWorks-VB and other software.
CompuServe: 70303,2252

**Michiel de Bruijn**
lives in Rotterdam, The Netherlands and is co-owner of VBD Automatiseringsdiensten, where he is the designer and lead programmer of the Network Communications System, a Windows-based messaging system written almost entirely in VB. In his spare time, he manages the *VBPJ* Forum's Localization section.
CompuServe: 100021,1061;
Internet: mdb@vbd.nl

**John D. Conley III**
is a senior computer consultant for Coopers & Lybrand. He is also a partner in New World Computing, an independent consulting firm. He lives in North Dallas, Texas.
Voice: 214-234-1137

**Gary Cornell**
has written 12 books on microcomputer technology, including *The Visual Basic 3 for Windows Handbook* (Osborne-McGraw Hill). He is a professor at the University of Connecticut and has been a visiting scientist at IBM's Thomas Watson Labs.
CompuServe: 75720,1524

**Craig Goren**
is president of Clarity Consulting Inc., a Chicago-based client/server consulting firm. He leads the client/server section of the *VBPJ* Forum on CompuServe (GO VBPJFORUM).
CompuServe: 72773,1062
Internet:cgoren@claritycnslt.com

**Deborah Kurata**
is a principal consultant and founder of InStep Technologies, a Pleasanton, California-based consulting group specializing in the design and development of Windows applications. She has written several articles for *VBPJ* and is working on a book covering VB application architecture. She is also the leader of the Beginner's Corner section of the *VBPJ* Forum.
CompuServe: 72157,475

**Blaine Leckett, Ph.D.**
is head of software development for QuantaVision Canada, which creates hardware and software for imaging analysis in scientific research. As a freelance programmer, he has written several Japanese-language translation and education programs in VB. He holds a Ph.D. in Experimental Medicine (Diabetes Research) from McGill University.
CompuServe: 72720,761
Internet: comcul@cam.org

**Ibrahim Y. Malluf**
is president of Malluf Consulting Services in Moriarty, New Mexico, a consulting and contract programming firm specializing in client/server, database, process control, and decision-support systems. He leads the Science/Industrial section of the *VBPJ* Forum.
CompuServe: 70661,1467
Internet: iymalluf@rt66.com

**MicroHelp Tech Support**
These tips were contributed by Bob Flickinger, Roy Taylor, Kelly Wiegard, and Adam Schmidt. MicroHelp Inc. is the maker of VBTools, HighEdit, and other tools for VB programmers.
Voice: 404-516-0899 or
800-922-3383

**Patrick O'Brien**
is cofounder of Siena Software Inc., a consulting firm based in Half Moon Bay, California that specializes in client/server systems. He leads the *VBPJ* Forum's Database Warehouse section and is chair of the Northern California Software Forum's client/server special-interest group.
CompuServe: 70713,3317

**Charles W. Peper**
is a programmer who lives and works in Oswego, New York
CompuServe: 73517,3574

**Karl Peterson**
is a GIS analyst for a regional planning agency and a member of *the Visual Basic Programmer's Journal* Technical Review Board. He's also an independent programming consultant and writer based in Vancouver, Washington.
CompuServe: 72302,3707

**Barry Seymour**
is a client/server consultant at DBSS Inc., an international company that specializes in client/server systems development. He is coauthor of *Using Visual Basic 3.0* (QUE) and is the leader of the *VBPJ* Forum's UI Studio section.
Voice: 415-583-3000
CompuServe: 70413,3405

**Sheridan Software Tech Support**
These tips were submitted by the technical support staff of Sheridan Software Systems, makers of 3-D Widgets, VBAssist, and other tools for VB programmers.
Voice: 516-753-0985
Fax: 516-753-3661

**Paul D. Sherriff**
is an independent consultant who specializes in VB, Access, and SQL Server applications and training. He is the author of *Visual Basic Power Guides.*
CompuServe: 72230,2216

**Mark Streger**
is a principal at Information Management Consultants in McLean, Virginia. He specializes in building client/server applications using VB.
CompuServe: 71700,3037

**William Storage**
is an independent software consultant who is based in both San Francisco and Boston. He is the leader of the *VBPJ* Forum's Code Style section.
CompuServe: 75250,1360

**Phil Weber**
is the founder of Micro Business Services, a contract programming firm based in Portland, Oregon, and a member of the *Visual Basic Programmer's Journal* Technical Review Board.
CompuServe: 72451,3401

**Jonathan Wood**
writes commercial and custom software using Visual Basic, Visual C++, and assembly language, and is a member of the *Visual Basic Programmer's Journal* Technical Review Board. His company, SoftCircuits, is based in Southern California.
CompuServe: 72134,263