Visual Basic Tips & Tricks

Visual Basic For DOS

- Files & Directories

Finding Directories

How do you test to see if a directory exists with Visual Basic for DOS? You might want to use the DIR$ command like this:

```
DIR$("c:\test\*.*")
```

This will work as long as there are files in the directory. But how about if the directory is empty? The above code won't work.

To get around this I use the "nul" specification. Every directory has a "nul" file in it, regardless if there are any other (real) files in it or not. Below is how to use it:

```
XY$ = DIR$("c:\test\nul")
IF XY$ <> "nul" THEN
   MKDIR "c:\test"
END IF
```

Submitted By: David McCarter

Visual Basic For Windows

- Access
- Buttons & Image Control
- Controls
- Forms
- Graphics
- Hot Spots
- List Boxes
- Menus
- Miscellaneous
- Text Boxes
- Windows

Access

- [ ] Printing Blank Fields

Printing Blank Fields

I was converting some Visual Basic 1.0 code to Visual Basic 3.0 using the Data Control to replace some database code. The problem was that when I would print an empty field to the printer (i.e. Printer.Print Data1.Recordset("Field1")) instead of the printer printing nothing it would print '**NULL**'. This was VERY frustrating. To get around this problem I wrote the following Sub Routine:

```
Sub PrinterPrint (something as Variant)
   If IsNull(something) = False Then
      Printer.Print something
        Else
           Printer.Print ""
        End If
End Sub
```

Then, using the DOS edit program I replaced all occurrences of 'Printer.Print' with 'PrinterPrint'.

Submitted By: Mike Payne

Buttons & Image Control

- Image Control As A Button
- Mouse Button Up Or Down Status

Image Control As A Button

One of the easiest techniques for adding graphical effects to your program is to use an image control as a button. When the image control receives a Click event, you simply substitute the value of the Picture property. The key to this technique is to define a pair of invisible image controls with pictures corresponding to the up and down status of the control.

For example, you could create a button that visually represents a locked and unlocked state. One advantage of using icon files rather than bitmap files is that any underlying image shows through the mask area of the icon.

When the form is loaded, the Form_Load event procedure sets the appropriate image in the image control:

```
Sub Form_Load()
   Padlock.Picture = LockOpen.Picture
End Sub
```

The image control responds to the click event by replacing the picture in the control:

```
Sub Padlock_Click()
Static LockedFlag As Integer
If LockedFlag Then
   Padlock.Picture = LockOpen.Picture
      Else
     Padlock.Picture = LockClosed.Picture
End If
LockedFlag = Not LockedFlag
End Sub
```

Source: Microsoft Developer Network News, July 1993

Mouse Button Up Or Down Status

Here is how to check the mouse button 'up_or_down' status.

```
Declare Function GetKeyState Lib "user" (ByVal k%) As Integer
Declare Function GetAsyncKeyState Lib "user" (ByVal k%) As Integer
```

This functions will tell you whether a key is up or down, and if it's "toggled". The high order bit tells you if the key is up or down. The low order bit tells you if it's toggled - each time the key is pressed and then released the low order bit will change.

The first function maintains the keyboard state when the last *window message* was received by the app and is the one you normally use. The second one corresponds to the real time keyboard state.

Now, why on earth do I talk keyboard when the question was about the mouse? - That's because the mouse buttons are considered as "virtual" keyboard keys. Some virtual key codes are:

```
VK_LBUTTON      1
VK_RBUTTON      2
VK_MBUTTON      4
VK_TAB          9
VK_ESCAPE      27
```

So to wait for the left mouse button to be first pressed and then released you could use:

```
Do While GetKeyState(VK_LBUTTON)>=0: DoEvents: Loop
Do While GetKeyState(VK_LBUTTON)<0: DoEvents: Loop
```

Submitted By: Dan Bystrom - InterNet:adbbyd@msmail.hk-r.se

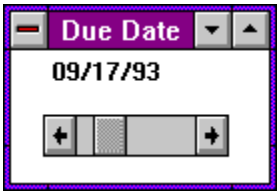
Controls

- Using Controls Indirectly To Change Data
- Copying Data From A Grid Control

Using Controls Indirectly To Change Data

In certain cases, you might not want the user to change data in a control directly, but indirectly using another control. This technique can minimize, or even eliminate the process of validating user input.

Let's say you want the user to enter a due date that is from 1 to 30 days from today's date. You can solve this scenario by using a horizontal scroll bar that modifies the Caption property of a label:



The scroll bar property settings are:

```
LargeChange=5
Min=1
Max=30
```

Every time you change the scroll bar value, the HScroll1_Change event procedure is generated, which in turn calls the DisplayDate procedure:

```
Sub HScroll1_ Change ()
   DisplayDate
End Sub

Sub DisplayDate ()
   TimeVal = Now + HScroll1.Value
   Label1.Caption = Format(timeVal, "mm/dd/yy")
End Sub
```

The TimeVal variable contains the current time returned from the Now function plus the current setting of the scroll bar's Value property (1-30). In Visual Basic, adding integer values to a time value increments the time value by days.

Lastly, to set the initial date value on the label, you need to call the DisplayDate procedure when the form is loaded:

```
Sub Form_Load ()
   DisplayDate
End Sub
```

Source: Microsoft Developer Network News, September 1993

Copying Data From A Grid Control

The grid control in Visual Basic provides the CLIP method for copying cells to the clipboard from the grid. The CLIP method puts a tab between each column and a line feed between each row. This works great for copying to Excel (I have not tried other spreadsheets), but does not work well copying into word processors because only the line feed is placed between each row, not line feed and carriage return. The Visual Basic manual describes how to insert the carriage return by copying the string one character at a time, inserting the carriage return in the appropriate places. Copying the grid with both line feed and carriage return allows pasting in Excel and word processors.

Looking at each character is time consuming. I have found that it is faster to copy each cell individually, inserting the tab and CR/LF characters as needed.

Below is the routine I use to copy an entire grid. Note that fixed rows and columns are not copied. To copy fixed rows and columns, you would set First_Col and First_Row to 0.

Note also that I set the MousePointer to 11 (Hourglass) at the begining and set it back to 0 (default) at the end. This is because large grids may still take a few seconds to copy on slower systems.

```
Sub CopyGrid_Click ()

    Dim CopyText, NC, NR As String
    Dim First_Col, First_Row As Integer

    screen.MousePointer = 11

    Clipboard.Clear

    NC = Chr$(9)
    NR = Chr$(13) & Chr$(10)

    First_Col = datagrid.Fixedcols
    First_Row = datagrid.FixedRows

    For i% = First_Row To datagrid.Rows - 1
        datagrid.Row = i%
        For j% = 0 + First_Col To datagrid.Cols - 1
        datagrid.Col = j%
            If j% = First_Col Then
                CopyText = CopyText & datagrid.Text
            Else
                CopyText = CopyText & NC & datagrid.Text
            End If
        Next
        CopyText = CopyText & NR
    Next

    Clipboard.SetText CopyText

    screen.MousePointer = 0

End Sub
```

Submitted By: Dennis Cabell

Forms

Detecting Previous Instances Of A Program

There are times when you may want to prevent a second instance of your program from running. The App object provides a PrevInstance property that allows you to determine whether a previous instance of the program is running. Here's how you might write your code:

```
Sub Form_Load ()
If App.PrevInstance Then
   msg$ = App.EXEName & " already running "
   MsgBox msg$, 48
   End
End If
End Sub
```

Notice that the procedure uses the EXEName property of the App object to display the program's name in the Visual Basic message box.

Source: Microsoft Developer Network News, July 1993

Portable Forms

The following gives a nice way to produce forms on one screen resolution while keeping the same aspect ratio on other resolutions.

While designing a form the programmer should choose the twips unit, and should always design using one fixed resolution (i.e. 800x600). What I mean by that is when a button is designed on an 800x600 screen it will become larger and maybe outside the screen area on a 640x480 screen.

To avoid such an annoyance we should read Screen.Width and Screen.Height during design time, and during run time. This will provide us with a ratio with respect to the reference at design time. This ratio is then used to readjust the form, shape, and/or button or all of the visible controls when the form is run. For example try:

```
Shape1.Width = Shape1.Width * Ratio
```

Where Ratio is Screen.Width / SCREEN_WIDTH_REFERENCE for the X coordinate, in a similar way this could be done for the Y coordinate.

Submitted By: Nadim El-fata

Setting The "TabIndex" Property

Here is a no-fuss method of setting the "TabIndex" property at design time.

One of the most enjoyable features of the Visual Basic environment is its ease of form design. Just plop some controls on a form, drag them around until you get the most appealing layout and you're done. Except of course you have to go back and set the "TabIndex" property for all the controls to match your final design.

My method used to be to click the first control, set its "TabIndex" to 0, click the next control set its "TabIndex" to 1, ... until I discovered this trick. Start with the last control (usually the bottom right) and work your way backwards (left and to the top). Each time you click a control set its "TabIndex" property to "0". When you reach the first control all "TabIndex" properties will now be in the correct order. This method works since VB references the "TabIndex" property for all controls on the form when you change the property to a value that already exists on another control. This method is even easier with the new properties window in VB3 because the "TabIndex" property will keep the focus even as you click controls. Just keep one finger on the "0" button and the other on the mouse. Then click, press "0", click, press "0", click press "0"...

Submitted By: Kyle Lutes

Graphics

- Decoding Binary CGM Graphics

Decoding Binary CGM Graphics

Here is how you decode binary CGM (Computer Graphics Metafile) files in Visual Basic 3.0 std. The binary CGM standard specifies byte-order and bit-order for multi-byte binary numbers as left-to-right (big-Endian), but the PC byte order is not left-to-right, it is little-Endian.

Byte-swapping is simple in C or C++, but not so simple in visual basic. (Problems with sign extension in signed integer types, and no unsigned integer types.) Thus, I used the following function to swap two-byte integers:

```
Function SwapBytes (num As Integer) As Integer
' Take an input integer, assumed to be in "left to right" byte order,
and convert it to "standard" Intel format by swapping the two bytes.

Dim TextVal As String
Dim NewTextVal As String
Dim StringLength As Integer

TextVal = Hex$(num)
StringLength = Len(TextVal)
Select Case StringLength
Case 1
   NewTextVal = "&H" & "0" & TextVal & "00"
Case 2
   NewTextVal = "&H" & TextVal & "00"
Case 3
   NewTextVal = "&H" & Right$(TextVal, 2) & "0" & Left$(TextVal, 1)
Case 4
   NewTextVal = "&H" & Right$(TextVal, 2) & Left$(TextVal, 2)
End Select
SwapBytes = Val(NewTextVal)
End Function
```

Submitted By: Steven W. Layten - InterNet: swl26@cas.org

This Topic Is Under Construction!

List Boxes

- Tab Stops In A List Box

Tab Stops In A List Box

The standard Visual Basic list box supports tab stops. This means that if the string value you add to the list box contains tab characters, the tabs cause the list box to display columns of strings.

Here's how to add the first item to the list box:

```
t=Chr$(9)
name="Michael Cage"
list1.AddItem name + t + "44" + t + "C/F"
```

You could also narrow the width of the list box so that you don't display the second and third columns. This technique allows you to store multiple strings per item, while only displaying the first string. Notice, however, that you would need to write additional code to extract specific strings.

Taking the idea of a list box as a storage mechanism one step further, you could make the list box invisible and only refer to it in your code.

Source: Microsoft Developer Network News, July 1993

Menus

- Pop-up Menus
- Help Menu On Right Side Of Menu Bar

Pop-up Menus

One of the new features of Visual Basic 3.0 is pop-up menus. You can easily create a pop-up menu from an existing menu structure. Let's look at how you would create a pop-up menu from the traditional menu in the Blanker sample application provided with Visual Basic (in your VB\SAMPLES\GRAPHICS directory).

First, use Menu Design windows to set the Visible property of mnuOption to False. Then, add the following event procedure to display the pop-up menu:

```
Sub Form_MouseUp (Button As Integer,...)
If Button = 2 Then
   PopupMenu mnuOption
End If
End Sub
```

Notice that the Form_MouseUp event procedure uses the right mouse button to display the menu.

Source: Microsoft Developer Network News, July 1993

Help Menu On Right Side Of Menu Bar

To position the Help Menu to the right side of the Menu Bar, use a backspace character (Chr$(8)) as the first character in the Caption property of the menu. You must do this at runtime, in the Form_Load event.

```
HelpMenu.Caption= Chr$(8) & HelpMenu.Caption
```

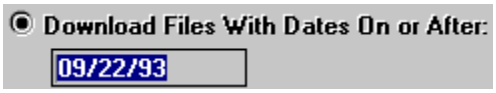
Tip By: Brian Dalton

Miscellaneous

- Automatic Selection Of Text
- Creating Screen Savers
- Easy Help
- Finding Hard/Floppy/Removable Drives
- Imitating A Combo Box
- Reading INI Files
- Re-reading The WIN.INI File
- Printer Control
- Setting TabIndex At Design Time
- Starting A Large Application
- Tool Box

Automatic Selection Of Text

When using text boxes, it is often useful to generate automatic selection of text when the control gets focus. You can do this easily by adding a couple of lines to the GotFocus event procedure:




```
Sub Text1_GotFocus ()
Text1.SelStart = 0
Text1.SelLength = 65000
End Sub
```

Notice that the length value for SelLength is 65000, nearly the maximum length allowed in a text box. This forces Visual Basic to use the actual length of the text as the SelLength.

Source Microsoft Developer Network News, July 1993

Creating Screen Savers

Overview:

Have you ever wanted to write your own screensaver, but was told, "You can't do that in Visual Basic!" In fact, one person I heard from told me that Microsoft said that screensavers were impossible in Visual Basic. I didn't believe hem, and this is the result.

Windows screensavers are easy and fun to write in Visual Basic, but there are some very important things to know.

Experts can go straight to the Code Section of this document.

Description:

Application Title:

A Windows screensaver is nothing more that a regular Windows executable file that has been renamed with the .scr extension. In Visual Basic, when you are making the executable file, you will need to set the Application Title in the Make EXE dialog box. This Application Title MUST be set to "SCRNSAVE title", where title is the text you want displayed in the Control Panel screensaver dropdown box.

Command Line Arguments:

When Windows starts up a screensaver it calls it with the "/s" argument, and when it wants to Setup the screensaver it uses the "/c" argument. So, we use a code module called SCRNSAV.BAS. In SCRNSAV.BAS you will see that a Select Case statement is used to capture this argument. You will need to change the Startup Form in the options|Project Dialog Box to Sub Main.

Telling Windows that the Saver is Running:

How long Windows waits before loading the screensaver is specified in the Control Panel. But if your screensaver doesn't tell Windows that it is running, Windows will reload the screensaver after that time passes again, even though the screensaver is already running. At first I thought I could remedy this situation by using VB 3.0's App object. The App.PrevInstance property will tell you whether or not there is a previous instance loaded.

This should've worked, and I got many comments saying that I must have messed something up, but it didn't. For some reason, with the screensaver this kills both instances, not just the second. But there is a way out. To fix this I found a Windows API call which tells Windows that the screensaver is inactive, so don't load one. To use this API you need to use the API call SystemParametersInfo. This function is used to change system wide parameters, such as whether or not the screensaver is active. Be careful when using this call, since changes are permanent. You will need to make sure that your screensaver turns the screensaver back off when it has ended.

See Sub Main, and Sub ExitNice in the Code Section.

Hiding The Cursor:

When you write a screensaver, you'll want it to hide the mouse cursor as well as whatever else your saver does. To do this you need to use the API call - ShowCursor. When ShowCursor(False) is called, the cursor is hidden; when ShowCursor(True) is called, the cursor is re-displayed. The Windows cursor is a shared object, so if your process hides it your process needs to redisplay it as well. See Code section.

Knowing when to end:

When your screensaver ends is up to you, but generally you'll want it to end if any of the following occur: mouse moves, button pressed, key pressed. To do this you will need to call a routine to exit properly from each of these

events in your screensaver form. See SaverForm. You need to call this routine because this is where the other half of the SystemParametersInfo call is made. If this is left out, the screensaver won't run again after it wakes up. Another problem is that the MouseMove message is sent if the cursor is over the form, REGARDLESS if it is moved or not. So, you need to check to see if it has moved somehow. See the Code Section for my solution. (Not necessarily the prettiest.

Code:

ScrnSave.Bas

```
declarations
    DefInt A-Z
    Const SWP_NOSIZE = 1
    Const SWP_NOMOVE = 2
    Const SPI_SETSCREENSAVEACTIVE = 17
    Declare Function ShowCursor Lib "User" (ByVal bShow As Integer) As
Integer
    Declare Sub SetWindowPos Lib "User" (ByVal hWnd, ByVal After, ByVal
X,
      ByVal Y, ByVal cx, ByVal cy, ByVal Flags)
    Declare Function SystemParametersInfo Lib "User" (ByVal uAction,
ByVal
      uParam, lpvParam As Any, ByVal fuWinIni)

  Sub Main
    Select Case Command$
      Case "/s", "/S"
        Res = SystemParametersInfo(SPI_SETSCREENSAVEACTIVE, 0, ByVal 0&,
0)
        Load SaverForm
        SetWindowPos SaverForm.hWnd, -1, 0, 0, 0, 0, SWP_NOMOVE Or
SWP_NOSIZE
        ok = DoEvents()
      Case "/c", "/C"
        ConfigForm.Show
    End Select
  End Sub
```

You may also need to add some initialization code for whatever your screensaver does.

```
Sub ExitNice
    Res = ShowCursor(True)  'Turn the cursor back on
    'reset screensaver
    Res = SystemParametersInfo(SPI_SETSCREENSAVEACTIVE, 1, ByVal 0&, 0)
    End
  End Sub

Saver.Frm
  Form_Load
    WindowState = 2          'Maximize the screensaver
    Me.Show            'Show the form
    This = ShowCursor(False) 'Hide the cursor

  Form_MouseMove
    If (OldX = 0) And (OldY = 0) Then
      OldX = X
      OldY = Y
      Exit Sub
    End If
```

```
    If (OldX <> X) Or (OldY <> Y) Then
      ExitNice
    Else
      OldX = 0
      OldX = 0
    End If

Form_Click
  ExitNice

Form_MouseDown
  ExitNice

Form_KeyDown
  ExitNice

Form_KeyPress
  ExitNice
```

Config.Frm

Windows will pass the "/c" argument to Sub Main if the "Setup" option is chosen from control panel. Here you can setup specific options for your screensaver. You might want to save these options in a .ini file (win.ini or your own). Its up to you! If your Config.Frm has a "Test" feature which starts the screensaver from the Config form, then you will need to be careful about remembering to turn on the cursor after the screensaver starts, and then turn it off before it ends.

Sources:

Conger, James L.., The Wait Group's Windows API Bible: The Definitive Programmer's Reference. The Wait Group: 1993. ISBN 1-878739-15-8
VBZ: The Electronic Journal on Visual Basic. Copyright 1993 User Friendly, Inc. Issue 01: January/February 1993

Disclaimer/Distribution:

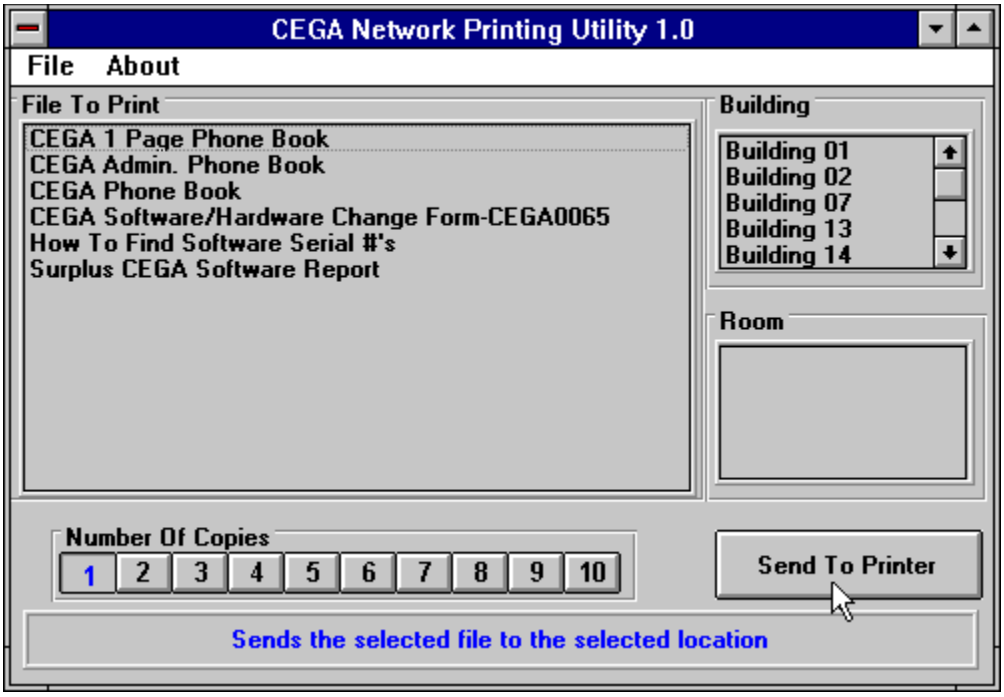
Submitted By: Peter Provost (provost@cs.colostate.edu)

Easy Help

I created a simple VB for Windows program that print company files to any network printer. I designed it to be easy to operate. When I went to write the help file, I decided against it. This would mean I would have to provide an additional file with the program and the time involved to write the help file was too much. I decided to add a text box with one line of help for each button.

I started by creating a SSPanel using the THREED.VBX file (any type of text box will do just fine). Then using the MouseMove event I added the following lines:

```
Sub Print_Now_MouseMove (Button As Integer, Shift As Integer, X As
Single, Y As Single)
HelpText = "Sends the selected file to the selected location"
End Sub
```



It's that simple! You can also use MouseDown, MouseUp, GotFocus events. I found the MouseMove the easiest to use. Just make sure when you code your program that all the buttons and the form have some kind of help text otherwise the mouse could be over a button and display the wrong help line.

Submitted By: David McCarter

Finding Hard/Floppy/Removable Drives

If you need to find what floppy drives, hard drives or removable drives (CD-ROM/Network) there is on a system, declare the following in a BAS file:

```
Declare Function GETDRIVETYPE Lib "Kernel" (ByVal nDrive As Integer) As Integer
```

You must pass this function the drive number, not a drive letter. Drive numbers always start a 0. For instance, drive A: is drive #0, Drive C: is drive #2.

```
DRIVETYPE = GETDRIVETYPE(DRIVENUM)
```

This function returns a 2 for removable drives, 3 for fixed drives or 4 for remote drives. This function will return a 0 if it is passed a drive number that does not exist on the system.

Submitted By: David McCarter

Imitating A Combo Box

To imitate a ComboBox that behaves like the one that appears in the "Search" option of Help, use this tip.

First, create a Textbox (eg. Text1) and a Listbox (eg. List1). Then paste the following code into the Declarations section of a form that contains the textbox and the listbox, and the bottom part of the code into a separate module.

Note the flags (in the declaration part of the module) that make this code not reentrant: it only permits one "combo" at a time (though many per application), but you can change this easily.

Code for module:

```
Option Explicit

Global CHClickList As Integer
Global CHInChange  As Integer

Sub CtrlTB_Change (OTB As TextBox, OLB As ListBox)
    Dim Pos As Integer, I As Integer, L As Integer
    Dim Aux As String

    If CHClickList Then
        CHClickList = False
        Exit Sub
    End If

    Aux = OTB.Text

    L = Len(Aux)
    For I = 0 To (OLB.ListCount - 2)
        If Not StrComp(Aux, Left$(OLB.List(I), L), 1) > 0 Then
            Exit For
        End If
    Next I

    OLB.TopIndex = I
    OLB.ListIndex = I
End Sub

Sub CtrlTB_KeyPress (OTB As TextBox, OLB As ListBox, KeyAscii As
Integer)
    If KeyAscii = 13 Then
        OTB.Text = Left$(OLB.List(OLB.ListIndex), 60)
        CHInChange = False
    Else
        CHInChange = True
    End If
End Sub

Sub CtrlLB_Click (OTB As TextBox, OLB As ListBox)
    If Not CHInChange Then
        OTB.Text = Left$(OLB.List(OLB.ListIndex), 60)
    Else
        CHInChange = False
    End If
End Sub
```

```
Sub CtrlLB_MouseDown ()
    CHClickList = True
End Sub
```

Code:

```
Sub List1_Click ()
   CtrlLB_Click Text1, List1
End Sub

Sub List1_MouseDown (Button As Integer, Shift As Integer, X As Single, Y
As Single)
   CtrlLB_MouseDown
End Sub

Sub Text1_Change ()
    CtrlTB_Change Text1, List1
End Sub

Sub Text1_KeyPress (KeyAscii As Integer)
   CtrlTB_KeyPress Text1, List1, KeyAscii
End Sub
```

Submitted By: Hernan Martinez Foffani, Buenos Aires - Argentina: email - hernan@condor.satlink.net

Reading INI Files

This is the subroutine I use to read INI files.

```
Function GetFromINI (SectionHeader$, VarName$, FileName$) As String
   Dim RetStr As String
   RetStr = String(255, Chr(0))
  'Get Requested Information
   GetFromINI = Left(RetStr, GetPrivateProfileString(SectionHeader$,
ByVal
   VarName$, "", RetStr, Len(RetStr), FileName$))
End Function
```

Of course declare the following:

```
Declare Function GetPrivateProfileString Lib "Kernel" (ByVal
lpApplicationName As String, lpKeyName As Any, ByVal lpDefault As
String, ByVal lpReturnedString As String, ByVal nSize As Integer, ByVal
lpFileName As String) As Integer
```

So you'd call it using something like:

```
Result = GetFromINI(Section, Variable name, Filename)
```

Submitted By: Daniel Bowen, Melbourne, Australia

Re-reading The WIN.INI File

To notify other apps on WIN.INI file changes, do:

```
SendMessage( HWND_BROADCAST, WM_WININICHANGE, 0, 0 )

HWND_BROADCAST = &HFFFF
WM_WININICHANGE = &H001A
```

Submitted By: James Shields

Printer Control

A user can control *everything* (printer, paper size, orientation, print quality, etc.) from within the print setup common dialog. In fact, here is all the code you need:

```
Sub FilePrintSetup_Click ()
Dim Msg$

    On Error GoTo FilePrintSetupError

    CMDialog1.Flags = PD_PRINTSETUP     ' Just enable print setup
    CMDialog1.Action = DLG_PRINT        ' Show printer setup dialog box

PrintSetupExit:
    Exit Sub

FilePrintSetupError:
    If Err <> CDERR_CANCEL Then         ' If user didn't select CANCEL
        Beep
        Msg$ = "Error" & Str$(Err) & ": " & Error$
        MsgBox Msg$, MB_ICONEXCLAMATION, "Error"
    End If
    Resume PrintSetupExit

End Sub
```

The capitalized constants are defined in CONSTANT.TXT. You do need t to set the CancelError property to True if you want the CDERR_CANCEL error to indicate the user pressed Cancel. (Even this really isn't needed, though).

Submitted By: Stephen C. Smith

Starting A Large Application

If you have a large Visual Basic application, it can take a while to load. The user sometimes may think that the application has failed to load when infact it has not yet loaded (or failed to load!).

Therefore, a good idea may be to have a small Visual Basic program that starts another much larger one.

To do this one has to use the SHELL command. But, this in itself is not enough, since the 'loading' form will not display before the shelled app. starts.

This is solved easily by the use of a Timer, which delays the SHELL command, thus allowing the starting form to display - with a suitable message, eg; "Loading Application, please wait..."

1. Create a form (herein called RunForm) with a Timer, a suitable message, and attributes as below, or as necessary.

```
ClipControls = 0   'False
   ControlBox = 0   'False
   MaxButton = 0   'False
   MinButton = 0   'False
   Tag = "RunForm"

   Timer Timer1
      Interval = 20  'Alter interval if req.
```

2. For the timer event, add the following code:

```
Sub Timer1_Timer ()

   Static NextOne

      If Not NextOne Then
         X = Shell(Filename, 1)
         Unload RunForm
      End If

      NextOne = Not NextOne

   End Sub
```

Submitted By: Darren Newbold > E-Mail: cdn@dmu.ac.uk

Tool Box

To create a tool box for your application, simply set up a form as a parent and another form as your toolbox/floating dialog whatever. In a suitable declarations section declare the API function as follows:

```
Declare Function SetParent% Lib "User" (ByVal hWndChild%, ByVal
hWndNewParent%)
```

For a floating toolbox over a parent form, try the following in the routine to show the toolbox:

```
Sub ShowTbox_Click ()
Dim ret As Integer
If doshow = False Then 'toolbox not visible
   ret = SetParent(tbox.hWnd, parent.hWnd) 'this makes the toolbox float

   tbox.Left = 0 'sets position to top left corner of parent
   tbox.Top = 0
   tbox.Show 'makes toolbox visible
   'try tbox.show 1 i.e. modal to see what happens
   doshow = True
   Showtbox.Caption = "&Hide Toolbox"
      Else
         tbox.Hide
      doshow = False
      Showtbox.Caption = "&Show Toolbox"
End If
End Sub
```

A couple of small caveats however. If you try tbox.show 1 i.e. modal you'll find the form will show but you will be unable to do anything with it. Secondly you absolutely *MUST* unload the child form i.e. tbox BEFORE unloading the main form otherwise your program will crash.

Submitted By: Matthew Dexter - InterNet:ch01md@surrey.ac.uk

Text Boxes

- Masking Input
- Horizontal Scrollbars
- Ignoring Keyboard Input

Masking Input

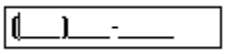
For greater control of data input, you might consider using the masked edit control. This control is included with the Visual Basic Professional Edition, along with many other custom controls.

The masked edit control provides a Mask property that lets you specify an input mask of literals and place holders. Literals provide visual cues about the type of data being used - for example, parentheses indicate a telephone area code. Placeholders represent a specific type of required value - for example, the # symbol indicates that you can only enter a decimal value (0-9).

Here's how you might set the Mask property to indicate a telephone number:

```
Mask        (###)###-####
```

At run time, the insertion point moves to the first placeholder:



The underscores in the control represent characters to enter. In this case, you can enter only numeric characters.

Source: Microsoft Developer Network News, September 1993

Horizontal Scrollbars

The reson you put a horzontal scrollbar on a textbox is usually because you want your text to be wider than the textbox control itself. However, if you set the horizontal scrollbar property to true on a multiline textbox, the text will no longer automatically wrap. The way I work around this is to set the with of the rect (EM_SETRECT) to my desired width when the form loads. I don't use the scroll bars property in the textbox. Instead, I place a horizontal scrollbar control below my textbox. I create my own sub that scrolls the text left and right by moving the rect (not changing size). I monitor the KeyUp and MouseUp events using the GetCaretPos function: I call the sub when required or if the scrollbar is changed.

```
Sub Form1_load (...etc...)
   Dim rtn&
   Dim Posn As Rect
   rtn& = SendMessage(Text1.hWnd, EM_GETRECT, 0, Posn)
   'get original rect
   Posn.right = NewWidth%
   'formatting width you require
   rtn& = SendMessage(Text1.hWnd, EM_SETRECT, 0, Posn)
   'set new rect width
End Sub
```

```
Sub ScrollRect (ByVal hWnd%, direction%)
   Dim rtn&
   Dim Posn As Rect
   rtn& = Send Message (hWnd%, EM_GETRECT, 0, Posn)
   'get current rect

   'to move rect left direction% is a negative number
   'to move rect right direction% is a positive number
   Posn.left = Posn.left + direction%
   Posn.right = Posn.right + direction%
   rtn& = SendMessage(hWnd%, EM_SETRECT, 0, Posn)
   'move rect left or right
End Sub
```

Tip By: Douglas Marquardt

Ignoring Keyboard Input

Here's an easy way to have your application ignore all input from the keyboard except for the Return and Backspace keys. (I wrote this code while creating a timer application that required text boxes that would accept only numbers.)

```
Sub Text1_KeyPress (KeyAscii As Integer)
   If KeyAscii = 13 Then
      KeyAscii = 0 'Prevents the system from beeping
      'Code or Sub to execute at Return
   End If
   If KeyAscii = 8 Then Exit Sub   'Allow use of backspace key
   If KeyAscii < 48 Or KeyAscii > 57 Then KeyAscii = 0   'Ignore other
keys
End Sub
```

This technique can be used to include ore exclude any characters based on their values in the ASCII character set.

Tip By: Daniel Switkin

Windows

- Keeping A Window On Top

Keeping A Window On Top

To keep a program window on top (always visible) in Visual Basic use a WINAPI function.

Code in Main Module:

```
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)
```

Code in a Submodule:

```
SetWindowPos form1.hWnd, -1, 0, 0, 0, 0, &H50 'This will make the window
always visible!
```

Code in Submodule 2:

```
SetWindowPos form1.hWnd, -2, 0, 0, 0, 0, &H50 'This will put "Always
Visible" off!
```

Submitted By: Henk Hakvoort

Windows 3.1 Help Files

- Adding Sound To A Help File
- Calling WINHELP
- Creating Bullets

Adding Sound To A Help File

To add sound support for .wav files in a windows help file:

In the [CONFIG] section of the project file:

```
RegisterRoutine("mmsystem","sndPlaySound","Si")
```

To call the DLL in a macro as a hotspot:

```
!sndPlaySound("anything.wav",0)
```

This is a way cool to way to build documents. It *could* be easier though.

Submitted By: Tod Massa - InterNet: MASSATR@SLUVCA.SLU.EDU

Calling WINHELP

Listed below are keywords used by WINHELP and an example on how to use it.

```
'Help engine declarations.
'Commands to pass WinHelp()
Global Const HELP_CONTEXT = &H1 ' Display topic identified by number in
Data
Global Const HELP_QUIT = &H2 ' Terminate help
Global Const HELP_INDEX = &H3 ' Display index
Global Const HELP_HELPONHELP = &H4 ' Display help on using help
Global Const HELP_SETINDEX = &H5 ' Set an alternate Index for help file
with more than one index
Global Const HELP_KEY = &H101 ' Display topic for keyword in Data
Global Const HELP_COMMAND = &H102 ' Execute Help macro
Global Const HELP_MULTIKEY = &H201  ' Lookup keyword in alternate table
and display topic

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

Type MULTIKEYHELP
     mkSize As Integer
     mkKeylist As String * 1
     szKeyphrase As String * 253
End Type

Case 2 ' Help Search Command
     HelpCmd = HELP_COMMAND ' Set help command to bring up
search box
     curFile = App.HelpFile
     curData = "Search()" ' Data is macro to be executed by WinHelp

' To make this work, we need to do two help commands in succession, so
we'll do one here and one down below. The commented line 2nd below is
the command to be executed to bring up the search dialog.
result = WinHelp(hWnd, curFile, HELP_INDEX, 0&)
result = WinHelp(hWnd, curFile, HELP_COMMAND, ByVal "Search()")
```

Submitted By: Gary Ferguson - InterNet: GARYFE@MICROSOFT.COM

Creating Bullets

One night I spent at least 2 hours trying to figure out how to create bullet items in a Windows 3.1 Help File. I found some Microsoft documentation which in both places it showed how to do it wrong! Below is the way that I found that works when using the QDHelp shareware program to compile the RTF file:

```
/para \tx360 \li360 \fi-360
{\f1 \'B7} \tab
Configurable to use many different sizes of diskettes.
Can even use 3 1/2 1.4MB (2.7MB) floppies compressed with Stacker.
/endpara
```

The '\tx360' RTF command sets the first tab stop to 360. The '\li360' sets the left indent to 360 and the '\fi-360' sets the first line indent to -360 or 0 in this case. The '\f1' sets the font to #1, which should be the Symbol font. The '\'B7' is the hex code for the bullet character in the Symbol font. The '\tab' moves the first line of text to the tab stop created with '\tx360'.

Here is what it looks like:

- Configurable to use many different sizes of diskettes. Can even use 3 1/2 1.4MB (2.7MB) floppies compressed with Stacker.

Submitted By: David McCarter

Other Sources Of Help

Visual Basic for Windows: Tips & Techniques

This very informative Windows Help File is released monthly (I think) and contains all of Knowledge Base articles.

This is a list of the main topics: VB Programming Using Standard Controls, VB Programming Using Custom & Third-Party Controls, Optimization, Memory Management, & General VB Programming, Advanced VB Programming -- Networks, APIs, DLLs, Graphics, Data Access & VB Database Programming, VB Design Environment, Running VB Applications, General VB References & Documentation Corrections, VB Setup, Installation, CDK, Help Compiler, DDE, & OLE

You can get the latest files off the Microsoft BBS or thru InterNet. The compressed file name is VBKB.EXE. This file comes with no search capibilities. If you want to search for topics or text, download VBKB_FT.EXE. Even though this file is much larger than the other, the search function in very valuable.

Sample

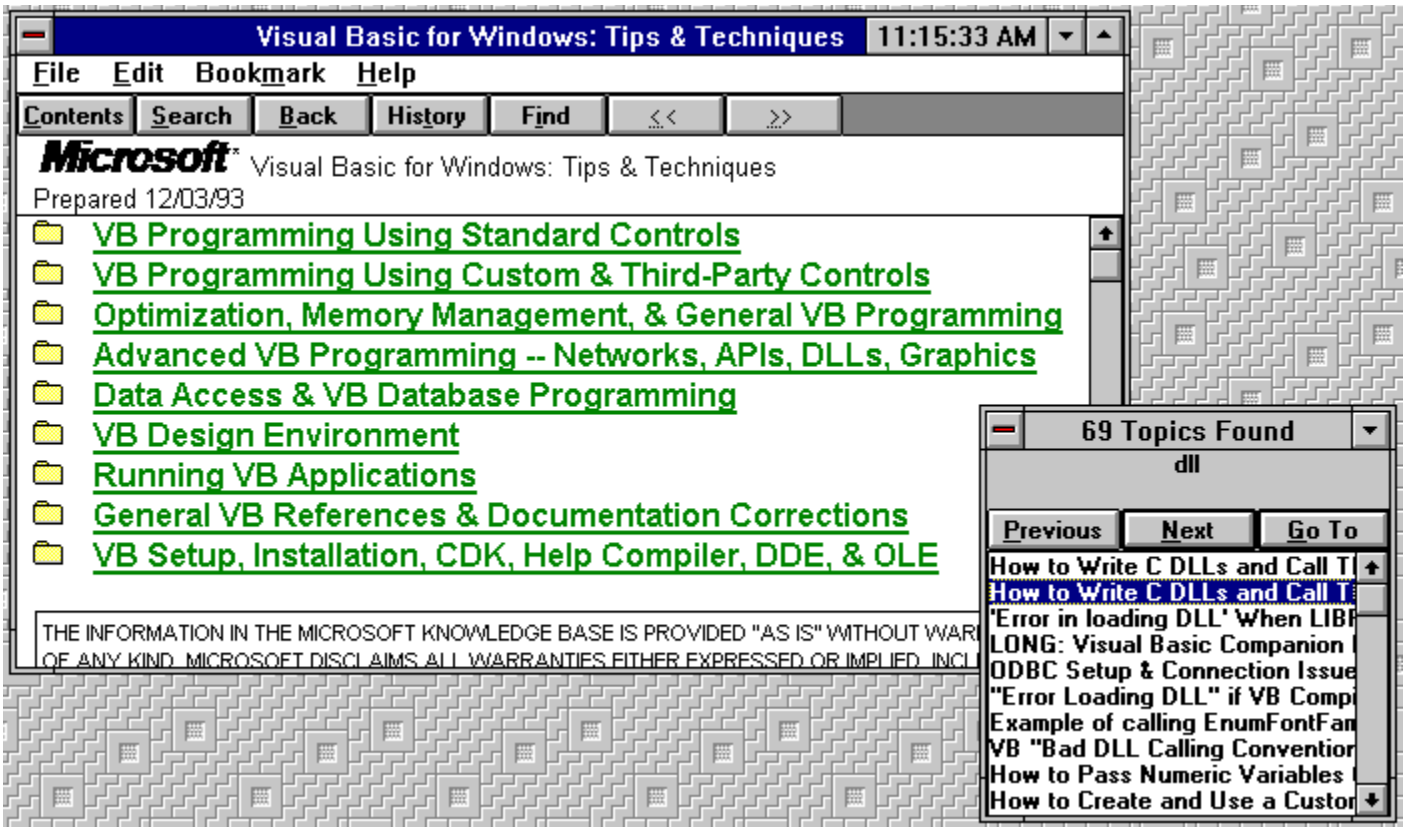
Visual Basic for Windows: Bugs, Fixes, & Updates

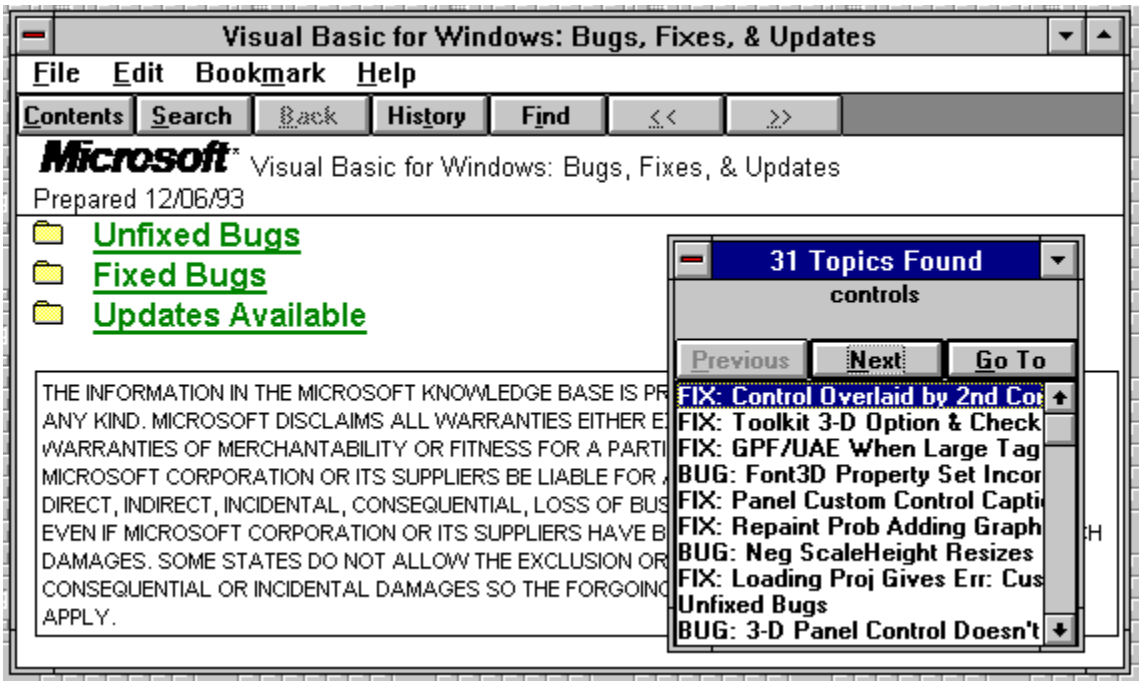
Instead of beating your head against the wall wondering why something won't work the way you think it should, first refer to this help file. I contains a list of unfixed bugs, fixed bugs (I guess that Microsoft knows about) and updates that are avaliable.

This is a list of the main topics: Unfixed Bugs, Fixed Bugs, Updates Available.

You can get the latest file off the Microsoft BBS or thru InterNet. This help file comes with VBKB_FT.EXE.

Sample


Visual Basic for Windows: Tips & Techniques
11:15:33 AM
File Edit Bookmark Help
Contents Search Back History Find << >>
Microsoft Visual Basic for Windows: Tips & Techniques
Prepared 12/03/93
VB Programming Using Standard Controls
VB Programming Using Custom & Third-Party Controls
Optimization, Memory Management, & General VB Programming
Advanced VB Programming -- Networks, APIs, DLLs, Graphics
Data Access & VB Database Programming
VB Design Environment
Running VB Applications
General VB References & Documentation Corrections
VB Setup, Installation, CDK, Help Compiler, DDE, & OLE
THE INFORMATION IN THE MICROSOFT KNOWLEDGE BASE IS PROVIDED "AS IS" WITHOUT WAR
OF ANY KIND. MICROSOFT DISCLAIMS ALL WARRANTIES EITHER EXPRESSED OR IMPLIED, INCL
69 Topics Found
dll
Previous Next Go To
How to Write C DLLs and Call T
How to Write C DLLs and Call T
'Error in loading DLL' When LIBI
LONG: Visual Basic Companion
ODBC Setup & Connection Issue
"Error Loading DLL" if VB Compi
Example of calling EnumFontFan
VB "Bad DLL Calling Convention
How to Pass Numeric Variables
How to Create and Use a Custor


Visual Basic for Windows: Bugs, Fixes, & Updates
File Edit Bookmark Help
Contents Search Back History Find << >>
Microsoft Visual Basic for Windows: Bugs, Fixes, & Updates
Prepared 12/06/93
Unfixed Bugs
Fixed Bugs
Updates Available
THE INFORMATION IN THE MICROSOFT KNOWLEDGE BASE IS PR
ANY KIND. MICROSOFT DISCLAIMS ALL WARRANTIES EITHER E
WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTI
MICROSOFT CORPORATION OR ITS SUPPLIERS BE LIABLE FOR
DIRECT, INDIRECT, INCIDENTAL, CONSEQUENTIAL, LOSS OF BUS
EVEN IF MICROSOFT CORPORATION OR ITS SUPPLIERS HAVE B
DAMAGES. SOME STATES DO NOT ALLOW THE EXCLUSION OR
CONSEQUENTIAL OR INCIDENTAL DAMAGES SO THE FORGOING
APPLY.
31 Topics Found
controls
Previous Next Go To
FIX: Control Overlaid by 2nd Co
FIX: Toolkit 3-D Option & Check
FIX: GPF/UAE When Large Tag
BUG: Font3D Property Set Incor
FIX: Panel Custom Control Capti
FIX: Repaint Prob Adding Graph
BUG: Neg ScaleHeight Resizes
FIX: Loading Proj Gives Err: Cus
Unfixed Bugs
BUG: 3-D Panel Control Doesn't

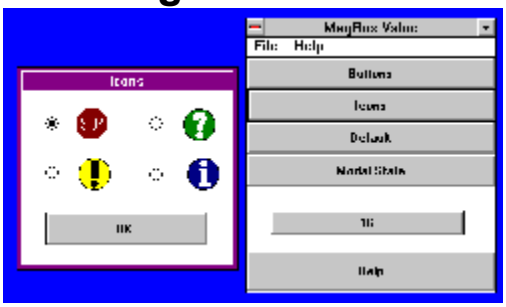
Programmer Tools

- Forms/Modules
- Programs
- VBX Files
- DLL Files

This purpose of this section is to let programmer know about Freeware VBX & DLL files and programs to make life easier for programmers. Developers of these types of programs are welcome to submit their program to us to be included with this help file.

Programmers Tool Programs

MessageBox Value



If you are like me, I can never remember all the different Message Box values available for use in Visual Basic for Windows. When ever I create a new message box, I always have to consult the Visual Basic help file. MessageBox Value takes care of this problem!

With MessageBox Value, you choose what buttons you want to use, which icon, which button is the default, task or system modal and MessageBox Value displays the Message Box value to use. You can even copy the value to the clipboard and paste it into Visual Basic.

This program is great for (as the developer puts it), "people with small memories",

MessageBox Value Is Developed By:Gavin Lazarow File Name: MSGVAL.ZIP

VB Program Updates

Updated On: 12/27/93

Visual Basic/Help Compilers

BTR110.EXE	71581	09-03-93	Updated Btrieve ISAM Driver Shipped W/Access
DATAINDX.EXE	36583	09-13-93	Index for the Data Access Guide -Professional Features Book 2
DATAMGR.EXE	38690	07-27-93	Source Code for Data Manager Tool
GENERIC.EXE	29705	07-27-93	CDK Sample left out of VBWIN 3.0
GRID.EXE	41357	07-27-93	Updated GRID.VBX, version 3.00.0538
HC505.EXE	228685	11-15-93	New Help Compiler for Use with WinWord 6.0 RTF File Format
HEALTH.EXE	250522	07-28-93	VBWIN 3.0 Pen Windows Sample Application
INSURE2.EXE	105250	07-28-93	VBWIN 3.0 Pen Windows Sample Application
LOANAPP.EXE	94661	07-28-93	VBWIN 3.0 Pen Windows Sample Application
MSAJT110.EXE	363656	10-20-93	Updated Access Engine Library, v 1.10.0001
MSCOMM.EXE	28231	07-27-93	Updated MSCOMM.VBX, version 2.1.0.1
ORA110.EXE	122816	09-02-93	Updated Oracle ODBC Driver Shipped W/ Access
RECEIPT.EXE	135573	07-28-93	VBWIN 3.0 Pen Winodws Sample Application
SETUPK.EXE	75847	10-20-93	Updated Setup Toolkit - SETUP.EXE, SETUP1.FRM, and SETUPWIZ.EXE
SQLUPDT.EXE	636209	09-13-93	Updated SQL & Oracle drivers
VBCOMDEM.EXE	30099	11-03-93	SIMPCOMM: Communications Demo Using Windows API's
VBKB.EXE	929182	12-06-93	Visual Basic Knowledge Base
VBKB_FT.EXE	2563620	12-06-93	VB KB Help Files w/ Full-Text Search
VBPRINT.EXE	95978	09-13-93	Sample Application and DLL that Demonstrate Advanced Printing in VB.
VBRUN100.EXE	172991	07-27-93	Copy of VBRUN100.DLL
VBRUN200.EXE	220504	07-27-93	Copy of VBRUN200.DLL
VBRUN300.EXE	246754	07-27-93	Updated copy of VBRUN300.DLL, v3.00.0538
XBS110.EXE	153916	07-27-93	Updated XBase driver, version 1.00.0002

Access

ACCBUG.TXT	29225	3-15-93	Bug list for MS Access. Contains known bugs and workarounds.
ACCFIX.TXT	34593	06-30-93	Known bugs that have been fixed in Access1.1
DDESRV.TXT	14341	07-08-93	Using Microsoft Access as a DDE Server
Q88164.TXT	6110	11-01-92	Intro to Windows Programming for MS-DOS Programmer
Q88173.TXT	2421	11-01-92	ODBC Setup for Access and SQL Server
Q88175.TXT	6026	11-01-92	Creating, Debugging, and Using an Access Library
Q88635.TXT	3268	07-28-93	How to Force a Cascading Delete
Q88653.TXT	4882	11-01-92	How to List the Related Tables in a Database
Q88655.TXT	7597	11-01-92	How Access Uses SQL Server Connections
Q88658.TXT	2622	11-01-92	Using TransferDatabase Macro to Attach to SQL Server Data
Q88907.TXT	4399	11-01-92	How to Modify the Toolbar in MS Access
Q88914.TXT	1023	11-01-92	Running MS-DOS SHARE with Windows for Workgroups
Q88940.TXT	6062	11-01-92	How to Dim (Gray) Menu Items with Access Basic
Q88999.TXT	1749	11-01-92	When to use Macros and When to use Access Basic
Q89590.TXT	1394	07-28-93	Making ENTER Add Lines in a Text Box
Q89594.TXT	3213	11-01-92	How to Display Immediate Window Without Module Window
WX0635.TXT	10260	09-10-93	Database Structure Questions & Answers

WX0636.TXT	5474	09-11-93	Access Basic and Macros Questions & Answers
WX0637.TXT	8147	09-10-93	Forms Questions & Answers
WX0638.TXT	8320	07-19-93	Nontechnical and Marketing Q& A
WX0639.TXT	9330	09-10-93	Querying Questions & Answers
WX0640.TXT	4661	09-11-93	Reports Questions & Answers
WX0811.TXT	11179	09-09-93	Interoperability with Other Applications Q&A
WX0812.TXT	12521	09-10-93	Setup Questions & Answers
WX0838.TXT	6423	06-20-93	Microsoft Access Questions & Answers
WX0867.TXT	9219	09-09-93	Access Distribution Kit Questions & Answers
4MEG.EXE	14402	1-08-93	performance enhancement suggestions for running Microsoft Access on a computer with 4 megs. of memory.
ACCADD.EXE	31601	02-12-93	Additional information on reported problems.
ACCSRV.EXE	23089	2-12-93	WinWord Template using Access as a DDE Server
BTR110.EXE	71581	12-07-93	Updated Btrieve Drive (Version 1.10.0011) for Access 1.1
CHOOSE.EXE	130760	05-25-93	This document addresses the breadth of database users and the unique challenges each group may face. It goes on to provide Microsoft's answer to these challenges in terms of appropriate database tools.
DDL100.EXE	45060	06-09-93	DLL to create & manipulate objects in AB. For Access V1.0 (Not Supported by PSS)
DDL110.EXE	38979	06-18-93	DLL to create & manipulate objects in AB. For Access V1.1 (Not Supported by PSS)
ERLIST.EXE	26417	2-12-93	List of Access Basic error messages
ORA110.EXE	122816	12-07-93	Updated ODBC Oracle Driver for Access 1.1
OUTPUT.EXE	206085	2-12-93	Saving report output to a file
PACKPT.EXE	50319	07-08-93	Updated version of Microsoft Object Packager
PRMPT.EXE	183011	05-12-93	Prompt.mdb sample-"Running Microsoft Access"
SECURE.EXE	37316	10-07-93	Additional Microsoft Access Security document
SECWIZ.EXE	211276	10-07-93	This wizard makes implementing database security and easy task with fixes for SYBASE SQL Server 4.8 and 4.9
SPT100.EXE	73938	06-09-93	SQL Pass-through DLL for Access 1.0
SPT110.EXE	69878	06-18-93	SQL Pass-through DLL for Access 1.1
SQL100.EXE	43338	06-14-93	INSTCAT.48 and INSTCAT.SQL Catalog Stored Procedure scripts v 1.0 for ODBC
SQL110.EXE	29171	06-14-93	INSTCAT.SQL Catalog Stored Procedure script for Access 1.1 and SQL.
TIMER.EXE	31404	2-12-93	TIMERDLL.DLL for creating a background timer
UTIL.EXE	53641	07-07-93	MS Access v1.1 UTILITY.MDA file
WX0928.EXE	2805269	10-06-93	Microsoft Access Knowledge Base Help File
WX0964.EXE	211148	12-08-93	SecurityWizard and White Paper

All these file are available from the MicroSoft BBS: 1-206-936-6735

Release History

Version 1.3 Released on: 1/3/94 - File Name: VBTIPS13.ZIP
Version 1.2 Released on: 11/29/93 - File Name: VBTIPS12.ZIP
Version 1.1 Released on: 10/1/93 - File Name: VBTIPS11.ZIP
Version 1.0 Released on: 9/9/93 - File Name: VBTIPS10.ZIP

About The Developer

The Visual Basic Tips & Tricks Help File
Is Compiled By: David McCarter

I can be reached at:
DPM Computer Solutions
8430-D Summerdale Road
San Diego, CA 92126-5415
USA.

InterNet-DPMCS@HIGH-COUNTRY.COM
Compuserve Users-Contact me using the address: INTERNET:DPMCS@HIGH-COUNTRY.COM

If I (or someone that I know) has tested the submitted code to make sure that it works, then we tell you by adding a graphic like the one below for the Visual Basic version it was tested.

Disclaimer: I will try to make sure that all the coding in this help file is correct and works! I am not responsible for any damage that might happen to your computer or programs. REMEMBER...save your work before trying any coding listed here.

This help file is freeware, but if you would like to make a donation, any amout will be accepted. Also, 10% of all donations will go to chairity, the rest will go to purchase programs (we will try to buy only shareware) to keep this project going. Please make checks payable to: David McCarter.

This Help File was written with the following shareware programs:

Visual Help V2.0e - Developed By: WinWare, P.O. Box 2923, Mission Viejo CA 92690. CompuServe Address: 70272,1656

About Visual Basic Tips & Tricks

The future of this project rests on your shoulders... I wanted to develop this help file to provide some sort of centralized forum for users to share VB Tips & Tricks. I want to pool information from many different sources into one help file. As you might already know, Microsoft books and help files are (in my opinion) not written very well. They do not contain much help that a normal programmer might want. Some of their information is even printed wrong! So where does one turn? To off-the-shelf books, user forums, magazine articles and the like. Try to keep all that organized to retrieve the information you need quickly. Not so easy, is it?

I want this help file to provide HELP with small VB Tips & Tricks. Undocumented ones, work arounds, easier way to do things etc... But I need your help!. I need your input!

Please submit your VB Tips & Tricks for others to use. Since this help file is freeware, the only thing you will receive is your name embedded in this help file forever.

How to submit your VB Tips & Tricks:

Write down your tip or trick explaining exactly what is it is. Provide any coding (make sure it works) and graphics if you want. Text must be is ASCII format and graphics must be 16 color in a BMP format.

Send your tip or trick to:

DPM Computer Solutions
c/o VB Tips & Tricks
8430-D Summerdale Road
San Diego, CA 92126-5415 USA

Please submit them on a 3 1/2 disk using a floppy disk mailer.

You can also save some $$$ and electronically send it to me at:

InterNet - DPMCS@HIGH-COUNTRY.COM
CompuServe - INTERNET:DPMCS@HIGH-COUNTRY.COM
Send any files using uuencoding.

How To Receive Updates:

The easiest way is to receive them electronically through e-mail. To get on the e-mailing list just submit a VB Tip or Trick! Use my e-mail address above and send me your name, e-mail address, coding preference (uuencode or binhex) and file size limitations your system might have long with your VB Tip or Trick. My system has a 500K limit, so if and when the zipped file reaches that size, e-mail will no longer work. This will also be posted on many InterNet systems and BBS systems to be listed below.

Where To Find New Issues:

BBS Systems:

The Centre (New Zealand): (09) 443-7679 (V22bis, V32, V42bis, MNP5)

Windows-R-Us (San Diego, CA): (619) 944-7368 2400/ 1200/ 300. (619) 944-8583 USR 14.4.

HighCountry East (San Diego, CA): 619-789-4391 USR Dual Standard V32bis, 619-788-0831 Compucom. San Diego callers use 619-440-0231

InterNet FTP Sites:

CICA: ftp.cica.indiana.edu [129.79.20.17]

GARBO: garbo.uwasa.fi

On-Line Services:
CompuServe: File name= VBTIPS.ZIP

Form_Load() Events

- Detecting Prevous Instances Of A Program

Setting TabIndex At Design Time

One of the most enjoyable features of the Visual Basic environment is its form-design facilities. Just plop some controls of a form, drag them around until you get the most appealing layout, and your done. Except, of course, you have to go back and set the TabIndex property for all the controls to match your final design.

My method used to be to click the first control and set its TabIndex to 0 (zero), click the next control and set its TabIndex to 1, and so on - until I discovered this trick: Start with the last control (usually the bottom right) and work your way backwards (left and to the top). Each time you click a control, set its TabIndex property to 0. When you reach the first control, all TabIndex properties will be in the correct order.

This method works because VB resequences the TabIndex property for all controls on the form when you change the property to a value that already exists on another control. With the properties windows in VB 3.0, the TabIndex property will keep the focus even as you click controls. Just keep one finger on the 0 button and another on the mouse. Then click, press 0, click, press 0, click, press 0, and so on.

Tip By: Kyle Lutes