Visual Basic ToolBox Pro.

Help for common Shareware & PD Visual Basic Addons

Save & Run

The most important function in VBToolBox is "Save & Run". In [design] or [break] mode when you press "Save & Run" VBToolBox will save all your work and place Visual Basic in [run] mode. Now, if your system crashes everything has been saved. I have included a Global HotKey {Shift + F9} for this function to make it easy and convenient to use regularly. Save & Run should be used in place of the F5 or Shift F5 commands. Guaranteed to save you hours of lost work and frustration!!!

Note:
If you havent given your project a name Save & Run will call the Save As dialog box.

Save Text / Load Text

As you design programs in Visual Basic, you will normally create controls, variables, and procedures during that design process that you will later delete. Visual Basic retains these unused variables and procedures. When you create the final exe file these unused pieces of code are also added, increasing its final size. On small programs this will not have much consequence, but on larger projects where many changes have been made over time, the added size of the final exe can lengthen the load time of your programs, and lengthen download times if you distribute your programs over BBS's. In order to remove these extra definitions from your final exes, you need to save all your event procedures on each form as a text file and then reload each one into its appropriate form. Then you must re-save your entire project before making a new exe file. VBToolBox automates the process of Save text/Load text and re-saving your project into this one command button. This process makes many API and Sendkey calls. On large projects this can take a little time so be patient. I've been using this procedure for quite a while on my machine and it has worked flawlessly, but, as always, please test this on a few of your own unimportant projects before relying on it.

Note:
If the Save text / Load text procedure ever stops and you get a message regarding the Global.bas file: Don't Worry! Visual Basic has a quirk with respect to the Global.bas file. This file is never created until you actually enter code into the module, or change the name from Global.bas to "MyProg.bas" etc. Since the default project.mak file lists a Global.bas the program will call the Save as Dialog Box and ask if you wish to save this file. Go ahead and save the file Global.bas, change the name if you wish, (this explicitly creates the file for that project) and then re-save the entire project. Then re-run Save text / Load text. This is just a quirk in Visual basic, not an error in this program, and nothing to worry about.

To maximize the benefits of this Save text Load text procedure, after the process is complete close down VB and restart before making your exe file. This insures that nothing in memory is included in your final exe.

Call Text Files

If you have programming examples, read.me files for third party custom controls, global constants and API function references that you use during your programming sessions, you can add these to the [Text File] section of the ToolBox2.cfg configuration file, and access them during your VB sessions. To Use Call Text: Click on the Text Button.(File Card Icon), a list box will appear on the left side. To choose from the list double click on any title, to remove the list dialog click with the right button.

Also see:

Configuration

Call Help

Visual Basics dedicated help file is called easily by pressing F1. Unfortunately if you have purchased any of the popular VB add-ons that include their own help files you must assign them to icons and return to the program manager to call them. By adding these files to the ToolBox2.cfg configuration file you can access them from within Visual Basic. To use Call Help: Click on the help button [Question Mark Icon], a list box will appear on the left side. To choose from the list double click on your choice. To remove the list dialog click with your right button.

Also see:

Configuration

Launch Programs

VBtoolBox allows you to launch any program from within Visual Basic. This feature allows you to stay within the programming environment and still have easy access to Icons, paint programs, text editors or any other program you may find necessary for developing your programs. In the configuration file "Toolbox2.cfg" add the title and location of programs you wish to access under the heading [Launch Programs]. Feel free to add as many as you wish, no limit.

```
[Launch Programs]
[Title$,              Location$]
Icon Works,           C:\Windows\Iconwrks.exe
Windows Paint,        C:\Windows\Pbrush.exe
VB Tips,              C:\vb\textfile\VB-Tips.exe
Call Help App,        C:\vb\hc\Callhelp.exe
Control hWnd,         C:\vb\vbfindid.exe
Notepad,              Notepad.exe
```

For a more detailed explanation of the configuration process also see:

Configuration

Vbx Files

VBtoolBox allows you to add Vbx files to your project with the click of your mouse. Click on the Vbx button and a list box will appear to the left containing all available Vbx files. Double click on any file with the left mouse button and it will be added to you project. A click on the list box with the right mouse button will close it. For this function to work properly enter the full path of the location of your Vbx files below the heading [Vbx Location] in the "Toolbox2.cfg" configuration file.

[Vbx Location]
C:\Windows\System

For a more detailed explanation of the configuration process also see:

Configuration

hWnd

In order to use the send message function you must have the handle of the program you wish to manipulate from within Visual Basic. This function will find and list all active apps running in windows and give you a task ID#, handle, and caption for each. The handle list box has two options: The default shows only visible windows and their handles, the second option also includes all invisible windows with captions. Don't save these handles as constants in your program, this list is for debugging purposes only. Use only to confirm the accuracy of your code. Handles are randomly assigned when an app. starts.

For a more detailed explanation of the configuration process also see:

Configuration

Configuration

The configuration file ToolBox2.cfg is a structured sequential file. You must use a text editor such as Notepad to make any changes to this file. I have include a call for this file using Notepad in the default configuration . Therefore you may edit your preferences from within VBToolBox. If you are unfamiliar with the term structured sequential file, you may think of it as a set of Quick Basics DATA statements with headings separating data for each function, and commas separating each variable. Do Not Change the headings or blank lines between function groups. VBToolBox uses these as keys in finding the correct variables. Under the Text File and Help File headings you will find a sub heading that contains a format example. After you finish editing this file you must restart VBToolBox for any changes to take place.

Printer Type:
[Not Registered]
Printer=HP Compatable

VBToolBox supports three basic printer types: HP Compatible, Dot Matrix, and Wide Dot Matrix. Enter your printer designation under the registration status heading [Not Registered]. For laser or ink jet printers that use standard sheet sizes and accept standard HP instructions, line 2 of the configuration file should read **Printer=HP Compatible.** This instruction enables both portrait and landscape printing capabilities. If you have a standard dot matrix printer that uses 9.5" perforated paper, line 2 should read **Printer=Dot Matrix**. This instruction disables landscape printing. If you have a wide carriage dot matrix printer, line 2 should read **Printer=Wide Dot Matrix.** This instruction also disables landscape printing, but resets the line instruction from 80 characters per line to 105 characters per line to take advantage of the increased paper width.

Printing Note:
Most printers that have Win 3.1 drivers should accept the HP API calls (Famous Last Words).
If your printer fails to accept these codes and you wish to try landscape printing use the **Printer=Wide Dot Matrix** designation to reset the line length and set your printer to landscape manually.

Some Dot matrix printers also accept the HP codes. If you feel the need to print sideways on your dot matrix, experiment away.

Below the headings [Vbx Location] enter the full path of the location of your Vbx files. VBToolBox uses this variable to find individual Vbx files.

[Vbx Location]
C:\Windows\System

VbtoolBox allows you to lauch programs from within Visual Basic
[Launch Programs]

[Title$,	Location$]
Icon Works,	C:\Windows\Iconwrks.exe
Windows Paint,	C:\Windows\Pbrush.exe
VB Tips,	C:\vb\textfile\VB-Tips.exe
Call Help App,	C:\vb\hc\Callhelp.exe
Control hWnd,	C:\vb\vbfindid.exe
Notepad,	Notepad.exe

Under the heading [Text Files], enter the variables for all text files you wish to access from within VB.
The sub-heading gives you the format for each line.
Tittle$ = Program name that will be entered into the VBToolBox list box.
Viewer$ = Program used to view this text file, (Notepad, Write, etc.)

Location$ = Location of text file.
DONT FORGET THE COMMAS Title$, Viewer$, Location$

[Text Files]

[Title$,	Viewer$,	Location$]
VBpro Setup Kit,	C:\windows\notepad,	C:\vb\setupkit\readme.txt
VBpro HugeArray,	C:\windows\notepad,	C:\vb\samples\hugearry\readme.txt
API Cardfile,	C:\windows\cardfile.exe,	C:\windows\wrkit\Function.crd
VB Constants,	C:\windows\Notepad,	C:\vb\texts\constant.txt
VBpro Constants,	C:\windows\Notepad,	C:\vb\texts\Const2.txt
Configure VBToolBox,	C:\windows\notepad,	c:\vb\toolbox1.cfg
VBpro Win API,	C:\dos\edit.com,	C:\vb\winapi\winapi.txt

Under the heading [Help Files] you will also find a sub-heading outlining the format of this section:
Title$ = Title of help file to be entered into the VBToolBox selection list box.
HelpFile$ = Path and name of Help file.
If your help files are located in your Windows\System directory all you need is the file name.
This function will only launch help files. Files that dont use the WinHelp.exe help engine should be entered in the Text File section.

[Help Files]

[Title$,	HelpFile$]
VB Knowledge,	C:\vb\vbknowlg.hlp
Control XRef,	C:\vb\ctrlref.hlp
Setup Kit,	C:\vb\setupkit\setupkit.hlp
Widgets #1,	SS3d.hlp
Widgets #2,	SS3d2.hlp
Widgets #3,	SS3d3.hlp
Win Api,	C:\vb\winapi\winsdk.hlp
API XRef,	C:\vb\winapi\apixref.hlp

*** End Config, VBToolbox ***

Important:

After you finish making changes to the Toolbox2.cfg file you must exit and restart VBToolbox for the changes to take effect!

ONE LAST NOTE:

1. **DON'T CHANGE THE HEADINGS OR SUB-HEADINGS, VBTOOLBOX USES THESE AS KEYS TO INSTALL THE CORRECT VARIABLES IN EACH FUNCTION.**
2. **LEAVE A BLANK LINE BETWEEN EACH FUNCTION GROUP.**
3. **YOU MAY ENTER AS MANY TEXT FILES OR HELP FILES AS YOU WISH UNDER THE RESPECTIVE SECTIONS.**
4. **DON'T FORGET THE COMMAS SEPARATING THE VARIABLES!!**

Correcting Errors:

Since VBToolBox is intended for programmers in the Visual Basic environment I assume this configuration file example is fairly easy to follow. Just substitute your file names and locations for the default examples and everything should work fine. (Famous Last Words!!). If I'm wrong send me some nasty E-mail on GEnie, Compuserve, or America Online and I'll add a configuration dialog in the next version.

In the Help File Function if you get the big blue error message check your entry for proper spelling and the correct path.

In the Text File Function, if you receive an error message from the text file viewer that you selected, check your Location$ entry for proper spelling and path. If you receive a load error message from VBToolBox, check your Viewer$ entry for spelling, path errors and a trailing comma. Check your Title$ entry, it must be 20 characters or less and must be followed by a comma.

If the number of entries in your Text File list box or Help File list box don't match the number you have entered into the configuration file YOU FORGOT A COMMA, OR YOU ADDED A BLANK LINE BETWEEN ENTRIES. All variables must be separated by a comma and all function groups must be separated by a blank line. No blank lines are allowed between sets of variables, only between function groups. Follow the default example...

If you have any comments or suggestions for future versions you can reach me on America Online, Compuserve, Or by writing to me at the address below.

America Online: HalJ 1

Compuserve 71042,1566

Hal Jurcik
7819 S. Vanport Ave
Whittier, CA. 90606

Ordering Info

Thanks for trying Visual Basic Toolbox Pro.

If you find VBToolbox Pro indispensable please register your copy and ensure future improvements:

Only $19.95 US funds
Make checks payable to:
Hal Jurcik
7819 Vanport Ave.
Whittier, CA. 90606

Contact me on COMPUSERVE or at the above address for information
on muti user site licenses.

To all that sent in suggestions I've tried to incorporate as many as I could. If you liked the first version I'm sure this new version will more than meet your expectations.

New Features:
Printing capabilities for everything from the smallest code fragment to your entire project.
Add VBx files to your project with the click of a mouse
List of all active windows tasks and their corresponding Task ID #;s and hWnd's
Launch any program from within Visual Basic
Help file
Ability to automatically save your project as a series of text files

---

Freeware & Shareware VB DLL's, VBx's and Programming Examples:

For all users who don't belong to an Online service and asked about availability of the various Freeware addons and text examples in my original default configuration example, I've put together a four disk set containing the very best of what's available.

You can find details and a list of what' available in the file "Addons.lst. " As long as there's interest I'll continue to update both the disk set, as new programs become available and this help file with programming examples.

DLL / VBx Reference

Disclaimer:

I've found these VBx & DLL files on par with their commercially available counterparts. They're all available Online or on the Shareware market. I've included the original authors documentation in Help file format as convenience for all of you out there like myself who are always trying to find the correct declarations somewhere in a pile of floppies.

As always if your not willing to take the time to test it first,
Don't complain IF... , or Don't use it!

Grid VBx
Diamond VBX
Multi Button VBX
Addons DLL
VBDos DLL
Ctlhwnd.DLL
LZH DLL
VB-Comm.VBx
Bubble Help DLL
MHelp VBX
Hotkey DLL
3D Frames DLL
Drag & Drop Dll

Code Examples & Tips

Disclaimer:

These are code examples I've found on various Online services. I've included them here as a convenience.

As always if your not willing to take the time to test it first,
Don't complain IF... , or Don't use it!

Copy Fix

Disclaimer:

This bug fix was posted on compuserve as a fix for the installation program included with the Professional Tool kit. I've included it here as a convenience.

As always if your not willing to take the time to test it first,
Don't complain IF... , or Don't use it!

This code fixes a bug in the CopyFile function in the SETUP1.BAS module in the Setup Kit distributed as part of the Visual BASIC Professional Tool kit. I note for the record the Setup Kit is part of a set of development tools sold by Microsoft. The kit contains source code which is intended to be used and modified by developers to build their own setup programs.

This submission reproduces only small fragments from one routine, for the purpose of allowing users to fix a bug in the routine. Because no copyright notice appears in any human-readable code in the setup kit, and because the portion reproduced is of no use to anyone who does not own the professional tool kit, this submission does not violate any copyright laws or CompuServe's policy with respect to posting.

Dan Smith, 6/10/92
Light Sciences, Inc.
639 Granite St.
Braintree, MA 02184

voice 617 849-8226
fax 617 849-0052

CIS 75105,1421
dpbsmith@world.std.com

This function failed copying a file of length 32350. In general it fails when the file length approaches 32767 mod 32768. The problem is that as written the function uses bad hygiene in conserving string space. The fixed function has been tested on files of size 32350, 32766, 32767, 32768, and 32769. Without the fix, 32350-32767 fail, 32768 and 32769 (as expected) succeed. With the fix, all succeed. What happens is the strings szBufSrc$ and szBufDest$ are created and never cleared, consuming a few hundred bytes;
then the statement

```
FileData = String$(LeftOver, 32)
```

consumes up to 32767 bytes;

then

```
FileData = String$(BlockSize, 32)
```

attempts to consume 32768 bytes, apparently BEFORE releasing the existing storage. At this point an Out Of String space error occurs, followed by an On Error Goto trap to ErrorCopy, followed by a No

Resume error when the function exits. (Thus the error is displayed as a No Resume error rather than an Out of String Space error).

The problem is that a few hundred + 32767 + 32768 > 65536. The key fix is to set FileData = "" just prior to FileData = String$(BlockSize, 32). This in itself fixes the problem. On general principles, I have added some other string-space-freeing statements and revised the error handling.

A quick-and-dirty fix is to reduce the BlockSize. 16384 works. With the fix, it is probably feasible to increase BlockSize above 32768, but why bother? Performance was never an issue.

Sugested Code Modifications:

```
Function CopyFile (ByVal SourcePath As String, ByVal DestinationPath As
String, ByVal filename As String, VerFlag As Integer)
    Dim Index As Integer
    Dim FileLength As Long
    Dim LeftOver As Long
    Dim FileData As String

.
.
.

>After these lines:

    If Not FileExists(SourcePath$ + filename$) Then
        MsgBox "Error occurred while attempting to copy file.  Could not
locate file: """ + SourcePath$ + filename$ + """", 64, "SETUP"
        GoTo ErrorCopy
    End If

>Make this change:

'DPBS 6/10/92
'Change ErrorCopy to ErrorCopy1.
'A plain GoTo (above) and an On Error Goto
'canNOT be handled at the same address because
'the error GoTo must be Resumed and the plain
'GoTo must NOT be.
'
'   Was:
'   On Error Goto ErrorCopy
    On Error GoTo ErrorCopy1    'DPBS

.
.
```

```
.
>After these lines
.
.
.
    If VerFlag Then
        szBufSrc$ = String$(255, 32)
        Call GetFileVersion(SourcePath$ + filename$, szBufSrc$,
Len(szBufSrc$))

        szBufDest$ = String$(255, 32)
        Call GetFileVersion(DestinationPath$ + filename$, szBufDest$,
Len(szBufDest$))

        If szBufSrc$ < szBufDest$ Then GoTo SkipCopy

    End If

>make this change

'DPBS 6/10/92
'Added two following statements
'It is interesting that statements like this were
'already placed at SkipCopy. It looks as if they should
'precede it.
'These statements by themselves allow
'32350 to copy, but not 32766 and 32767.
'I am leaving them in on the grounds of general good
'practice.

    szBufSrc$ = ""      'DPBS
    szBufDest$ = ""     'DPBS

    '-------------
    'Copy the file
    '-------------

'DPBS 6/10/92
'One acceptable quick-and-dirty way to fix the
'problem is simply to reduce this constant to 16384.

    Const BlockSize = 32768

    Open SourcePath$ + filename$ For Binary Access Read As #1

.
```

```
.
.
>After these lines:

    NumBlocks = FileLength \ BlockSize
    LeftOver = FileLength Mod BlockSize

>Make these changes:

'DPBS 6/10/92
'The following FileData = "" is put here on the basis
'of general principle; it is not actually needed for the
'fix.
    FileData = ""                       'DPBS
    FileData = String$(LeftOver, 32)

    Get #1, , FileData
    Put #2, , FileData

'DPBS 6/10/92
'The following FileData = "" is sufficient by itself
'to fix the problem.

    FileData = ""                       'DPBS
    FileData = String$(BlockSize, 32)

.
.
.
>After these lines:

SkipCopy:
    szBufSrc$ = ""
    szBufDest$ = ""
    Screen.Mousepointer = 0
    CopyFile = True
    Exit Function

>Make this change:

'DPBS 6/10/92
'Added next three statements. The old code
'properly handled an ANTICIPATED error (file not found)
'but did not properly handle the Out Of String Space error.

ErrorCopy1:
```

```
    MsgBox "Error" + Str$(Err) + Chr$(13) + Chr$(10) + Error$(Err)
    Resume ErrorCopy

ErrorCopy:
    CopyFile = False
    Close


End Function
```

Printing Code

The Visual Basic Toolbox has two separate printing functions:

Print a code fragment:

Even with the best monitors, reading code is difficult on screen. The Print clip function allows you to select text with the mouse and print only that fragment for easier analysis. Two methods are available for accessing this function. After selecting a block of text with the mouse press the "F6" function key. This hot key will call the active form you are working on and print the text you have highlighted. A button for this function is also available in the "Text Operations" dialog box.

All code formatting is retained. The form name, date, time and page number is included in the header for future reference. Text is wrapped automatically. Of course the printer must be ready for this function to work.

Printing text files:

The text operations button in VBToolBox has three functions. The printing capabilities are contained in the Create Text File operation. As each form is saved as a text file, an entry is made in a list box to the left of the VBToolbox. Once all files have been saved as text you may select any or all the files in the list box for printing. Click the list box with the right mouse button and choose Portrait or Landscape orientation to print the selected files.

Caution:

Creating a new set of text files will overwrite all older versions in that directory. If you wish to save an older version of your text files, either move them to another directory or floppy disk before creating the new set, or create the new set in a different directory.

Naming convention for "frm" and "bas" files:

DO NOT give the same name to a "bas" file and a "frm". Saving your project as text files assigns the standard "txt" extension to all files, therefore the second file will overwrite the first!!!

Notes on using landscape mode:

On my DeskJet 500 landscape mode slows down printing about 10 fold. Therefore if you have a large program be prepared to look at an hour glass cursor for a while. **Be patient**, the system hasn't locked up, It just takes a while to rotate all that text.

**The landscape function uses the code for HP printers. It's been tested on laserjets, deskjets, and dot matrix printers, but I can't guarantee compatibility with all models or compatibles. If you have the code for a printer that is not HP compatible send it to me and I'll try to incorporate it into a future version.

Saving Code as text files

The Save code as text file function gives you the ability to automatically save your project as a series of text files in any directory you choose. When you click on the Create Text File button a dialog box appears to allow selection of a directory. There is no default, you must explicitly choose a directory. When creating text files all previous versions in that directory will be overwritten, if you wish to save and compare different versions make sure to save to an alternate directory!!

Usage:

a) Text files make an easy backup of your projects, if your making major changes to a program you can use these files as a reference, or restore your project to its original version if desired.

b) Copy text files to floppy for circulation among extended work groups.

c) Even on the best monitors viewing code on screen is difficult. You can use VBtoolbox's printing functions to make hard copies of your project code for easier examination and circulation.

Diamond VBx

Diamond Arrow Pad for Visual Basic 1992 by Mark Gamber

This gives Visual Basic an arrow control consisting of an up, down, left and right button combined into four "sub buttons" in a diamond shaped pad. Add the file VBDIA.VBX to VB by selecting "Add File" in the "File" menu and selecting the file from the listbox. You should see a diamond shaped tool appear in the toolbox and, if so, you're ready to use the control.

The control provides the "Click" procedure which passes an integer value to your VB code named "Button". The Button values are:

0 - Top button clicked (up arrow)
1 - Right button clicked (right arrow)
2 - Bottom button clicked (down arrow)
3 - Left button clicked (left arrow)

Along with the control, there should be an example of how it may be used in Visual Basic. Examine the "Click" procedure for more information.

This is free software for anyone who wants it provided:

1. Credit is given where credit is due.
2. I'm not liable. End of story.

Questions, bugs and general praise may be sent E-Mail on America Online to MarkG85. No other correspondence will be answered, if received.

Multi Button VBx

Multi-Button Control for Visual Basic by Mark Gamber 1992

The Multi-Button is a single control which divides itself into several small "sub-buttons", each acting as an independent button in itself. The Multi-Button may be used for multiple choice entries, array and matrix entries, spreadsheet headers and so on. Each sub-button may be individually raised or pressed under software control and the raised and lowered background and text colors may be changed for an entire control. If you have the original version, this version far surpasses it's functionality as will be demonstrated. To use the button, copy VBMBTTN.VBX to your Visual Basic directory, select "Add File..." from the Visual Basic "File" menu and select VBMBTTN.VBX from the list of files. Once selected, a Multi-Button icon should appear in the toolbox.

Limitations:

The Multi-Button is capable of dividing into 256 sub-buttons. Should you go beyond that limit, you will be greeted with a UAE. There is no error checking provided to ensure you stay within that limit as it could be exceeded by any combination of things. Therefore, it is up to the programmer to ensure no more than 256 sub-buttons are displayed by a single Multi-Button.

Events:

Only one event is supported directly by the Multi-Button. "Clicked" provides the sub-button number and whether the button click raised or lowered the sub--button. "Button" always begins with 0 and runs to 255. The actual range to expect depends on the number of sub-buttons displayed. "Pressed" is 1 if the sub-button was pressed, 0 if it was raised by the click event.

GotFocus and LostFocus are supported by Visual Basic. Basically, since the control cannot use keyboard input, it should never get the focus. In the off chance that it does, you should use GotFocus to immediately change the input focus to another control on a form.

Properties:

The Multi-Button numbers it's sub-buttons internally from 0 to 255. You may change the number DISPLAYED by altering the value in "NumBase". NumBase is added to the internal sub-button count as the sub-buttons are painted. Thus, if you want a range from 1 to 10, you would set NumBase to 1 and size the control so 10 sub-buttons are displayed. Numbers aren't the only thing displayed. By selected "True" for the "Alpha" property, letters may be displayed. In the case of letter labelling, a NumBase of 0 will cause the first sub-button to display as the letter "A". The sub-buttons are incremented from "A" to "Z", "AA" to "AZ", "BA" to "BZ" and so on up to "ZZ". "Width" and "Height" are supported directly by the Multi-Button and are

available in pixel format only. They comprise the width and height of the entire control. To set or look up the size of a sub-button, use the "SubWidth" and "SubHeight" properties. They, too, are available only in pixel format. Note that changing the SubWidth or SubHeight values should be followed by a change in size to the entire control to prevent partial sub-buttons from

appearing. This is easily accomplished by setting the control size to the number of sub-buttons to be displayed times the Subsize changed. The sub-button background color may be set for both sub-button states. The default values are light gray for both states. In addition, the text color may be set for both conditions. Defaults are black if raised, blue if pressed. The

"BackUp" property sets and retrieves the raised color value and "BackDown" sets and retrieves the pressed color value of the sub-button background. "TextUp" and "TextDown" perform the same function for the sub-button text. Setting these properties affects ALL sub-buttons in a given control.

Methods:

Not being very accustomed to Visual Basic terminology, I am not too sure of the difference between a property and a method but this seems to fall more into the latter catagory.

A sub-button may be checked to see if it's raised or pressed. In addition, a sub-button may be raised or pressed under software control through the "Value" method. Since there may be up to 256 sub-buttons per Multi-Button, this is an array, each array part specifying a single sub-button. Pressing a sub-button consists of setting the Value for a sub-button to a non-zero number and raising a sub-button consists of setting a sub-button value to zero. Again, the array

ALWAYS starts a ZERO, regardless of the button number displayed which only reflects the NumBase value. For example, if your Multi-Button is displaying sub-buttons from 1 to 10 and you want sub-button 1 pressed, you would:

```
MultiBtn1.Value( 0 ) = 1
```

To see if the same button has been pressed:

```
if MultiBtn1.Value(0) <> 0 then ...
```

Additional information:

Thanks to PCC David for his suggestions leading to this version of the control. This is free software given two conditions:

1. Any information pertaining to the author (me) MAY NOT be changed.
2. I'm not liable. Period.

MBDEMO has been included along with the Visual Basic source for the program to serve as an example of Multi-Button use. Questions, bugs and complaints are welcome. E-Mail to PCA MarkG on America Online. No substitutes. I don't take phone calls, letters or BBS mail.

Mark Gamber (PCA MarkG)

Addons DLL

Visual Basic Add-on DLL Version 1.01

This Dynamic Link Library (DLL) provides the following additional functions for Visual Basic:

Huge Array Support - Support for arrays which exceed Visual Basic's limitations
Disk Information Support - Access to disk/diskette information not available through Visual Basic

Michael A. Stewart
539 Dutch Neck Road
East Windsor, NJ 08520

If you have any questions about its functions or suggestions for its future enhancement the author can be contacted through the following services:

Compuserve ID: 76234,3314
Prodigy ID: HWKS33A
AOL ID: Michae1334
ILINK Windows, Basic, and Win.App.Dev Conferences.

Complete details as to the use of this software may be found in the included VBADDONS.HLP file through the standard Microsoft Windows WINHELP.EXE program (run winhelp.exe vbaddons.hlp)
Addons DLL Declarations

VBDos DLL

These are the DLL declarations for the Diskstat.dll of VBDos.

```
Declare Function GetAvailDisk Lib "diskstat.dll" (ByVal diskno%) As Long
Declare Function GetDisksize Lib "diskstat.dll" (ByVal diskno%) As Long

Declare Function IsFileHidden Lib "diskstat.dll" (ByVal fname As String) As Integer
Declare Function IsFileRO Lib "diskstat.dll" (ByVal fname As String) As Integer
Declare Function IsFileArchive Lib "diskstat.dll" (ByVal fname As String) As Integer
Declare Function IsFileSystem Lib "diskstat.dll" (ByVal fname As String) As Integer
Declare Function SetFileHidden Lib "diskstat.dll" (ByVal fname As String, ByVal plusmin As String) As
Integer
Declare Function SetFileSystem Lib "diskstat.dll" (ByVal fname As String, ByVal plusmin As String) As
Integer
Declare Function SetFileRO Lib "diskstat.dll" (ByVal fname As String, ByVal plusmin As String) As
Integer
Declare Function SetFileArchive Lib "diskstat.dll" (ByVal fname As String, ByVal plusmin As String) As
Integer

Declare Function GetFileTime Lib "diskstat.dll" (ByVal fname As String, ByVal retval As String) As Integer
Declare Function SetFileTime Lib "diskstat.dll" (ByVal fname As String, ByVal newtime As String) As
Integer
Declare Function FindFile Lib "diskstat.dll" (ByVal fname As String, ByVal searchpath As String, ByVal
fullname As String) As Integer
```

CtlhWnd DLL

Here's the function declaration for the CtlhWnd.dll along with an example for using the function. Use this dll to find the handle of any VB control.

Declare Function ControlhWnd Lib "CTLHWND.DLL" (Ctl As Control)

Wnd = ControlhWnd(control)

LZH DLL

WLZHCXP.DLL Visual Basic and C Advanced File Function Library.
Ver 0.2 12/30/91

This library was primarily written for use with Visual Basic, although C programmers may use it just as easily. Although these functions could be done is pure Visual Basic, this library greatly increases the speed of the functions. Compression is achieved using the LZH algorithm. Although somewhat slower than implode, it can obtain much better compression than implode. (Implode is the main compression used by ZIP.) Compression is dependent on the amount of repetition in a file. If a file contains a large amount of the same values of data, compression should be very good. Text and pictures with relatively large areas of a single color are good examples. Conversely, a .COM file, usually very compact and not very repetious, are usually poorly compressed, sometimes by an insignificant amount. The tradeoff for the amount of compression achieved by the LZH algorithm is speed. It can take a while to compress a large file.

Library code was written for Microsoft C 6.00A. Data space consists of about 200 bytes. The data used for compression and decompression is dynamically allocated from the global heap. This means the calling program does not have to have a bloated data space and the data are may be marked as shared. Some code was written in assembler to prevent linking the standard C library, bloating the initial code segment to nearly 1000 bytes. As it stands, about 300 bytes in initially loaded, the rest only when called.

This library is free to use by one and all AS LONG AS you agree not to hold me liable for and damage to hardware, software, firmware or your mental health by using this software. By using the library, the agreement is made. Otherwise, I would suggest you throw it away and pretend it never existed.

Questions and suggestions made be made to me through the following by way of America Online, E-Mail to MarkG85.

This version corrects a bug in the DeleteFile() function where the DS and DX register contents were reversed during an error condition. In addition, a simple Visual Basic program serves as an example of how to use the library in that environment.

WLZHCXP.DLL is a Dynamic Linked Library (DLL) which provides additional functionality to a number of languages at very little cost to the program itself. GDI.EXE, USER.EXE and KRNL386.EXE are other examples of DLL files. To access this library, or any other, from Visual Basic, you must define the function names and the type of variables they expect to recieve and return. If the function definitions are incorrect, you are cheerfully told so by the smiling face of a UAE. Below are the definitions of functions available from this library and the types of variables they expect and return. Variables ending with a percent (%) sign are integers, those ending with an ampersand (&) are long integers and those ending in a dollar sign

($) are strings. A number of variables are filenames and thus require a string to be passed such as filename$ or "filename.ext". For more infor-mation, see the Visual Basic example provided. And speaking of which...

Examples are not my strong point and this one is no exception. To use the program, start by copying this file, WLZHCXP.TXT to your Visual Basic directory and press the "Compress" button which creates the file for the "Decompress" function. Next, run "Decompress" which creates the file for "Append", "Copy" and "Delete". After "Delete", you must again either "Copy"

or "Decompress" to create the file WLZHCXP.OUT. Programming is straight-forward and commented enough to figure out what's happening.

To use the library from Visual Basic, you must define the return constants and functions in your Global Module as such:

```
Global Const LZH_OK = 0
Global Const LZH_MEMERROR = 1
Global Const LZH_NOTLZH = 2
Global Const LZH_FILEERROR = 3

Global Const LZH_READONLY = 1
Global Const LZH_HIDDEN = 2
GLobal Const LZH_SYSTEM = 4

Declare Function Compress% LIB "wlzhcxp.dll" (ByVal InName$,ByVal
                                                OutName$)
Declare Function Decompress% LIB "wlzhcxp.dll" (ByVal InName$,ByVal
                                                OutName$)
Declare Function AppendFile% LIB "wlzhcxp.dll" (ByVal Dest$,ByVal
                                                File2$)
Declare Function DeleteFile% LIB "wlzhcxp.dll" (ByVal Fname$)
Declare Function GetFileSize& LIB "wlzhcxp.dll" (ByVal Fname$)
Declare Function CopyFile% LIB "wlzhcxp.dll" (ByVal Srce$,ByVal Dest$)
Declare Function GetFreeDiskSpace& LIB "wlzhcxp.dll" (ByVal Drv$)
Declare Function SetFileStatus% LIB "wlzhcxp.dll" (ByVal Fname$, ByVal
                                                   Status% )
```

Using the library from C is considerably more flexible. For the sake of and example, I'll use the automatic runtime linking method:

* SOURCE.H:

```
#define LZH_OK          0
#define LZH_MEMERROR   1
#define LZH_NOTLZH      2
#define LZH_FILEERROR 3

int FAR PASCAL Compress( LPSTR, LPSTR );
int FAR PASCAL Decompress( LPSTR, LPSTR );
int FAR PASCAL AppendFile( LPSTR, LPSTR );
int FAR PASCAL DeleteFile( LPSTR );
long FAR PASCAL GetFileSize( LPSTR );
int FAR PASCAL CopyFile( LPSTR, LPSTR );
```

* SOURCE.DEF

```
IMPORTS
    Compress=WLZHCXP.2
```

```
    Decompress=WLZHCXP.3
    AppendFile=WLZHCXP.5
    DeleteFile=WLZHCXP.6
    GetFileSize=WLZHCXP.4
    CopyFile=WLZHCXP.7
    GetFreeDiskSpace=WLZHCXP.8

*   SOURCE.MAK

link /NOD SOURCE,,,SOURCE.EXE,MLIBCEW LIBW WLZHCXP,SOURCE.DEF
```

Note that the library was compiled and linked using the MEDIUM memory model.

Error codes:

```
    LZH_OK:           Everything went OK, no errors.
    LZH_MEMERROR:   A memory error occurred. Unable to continue operation.
    LZH_NOTLZH:      File to be decompressed was not compressed.
    LZH_FILEERROR: A file error occurred. Unable to continue operation.
```

Function listing:

```
int FAR PASCAL Compress( LPSTR InName, LPSTR OutName )
    Compresses data taken from source file, writing results to destination.

    Parameters:
        InName:   Filename of source file to be compressed.
        OutName: Filename of destination file to hold compressed data.

    Returns:
        LZH_OK if no errors or error code listed above.

*---------------------------------------------------------------------------*

int FAR PASCAL Decompress( LPSTR InName, LPSTR OutName )
    Decompresses data from source file, writing results to destination.

    Parameters:
        InName:   Filename of source file containing compressed data.
        OutName: Filename of destination file to receive decompressed data.

    Returns:
        LZH_OK if no errors or error code listed above.

*---------------------------------------------------------------------------*
```

int FAR PASCAL AppendFile(LPSTR lpTarget, LPSTR lpFile2)
Adds a file to the back of another file.

Parameters:
lpTarget: First source filename. This file receives the second file.
lpFile2: Second source filename. This is added to the first file.

Returns:
LZH_OK if no errors or error code listed above.

Notes:
lpTarget MUST be a valid file. If the filename specified is not found, it is NOT created. The second file specified is added the end of the first, but is not deleted.

---

int FAR PASCAL DeleteFile(LPSTR Filename)
Deletes file(s) specified by filename. You may use wildcard characters.

Parameters:
Filename: Name of the file to be deleted from disk.

Returns:
LZH_OK if no errors or error code listed above.

---

long FAR PASCAL GetFileSize(LPSTR Filename)
Obtains the byte size of a file.

Parameters:
Filename: Name of file to find the size of.

Returns:
0 if file not found or file size as a 4 byte long value.

---

int FAR PASCAL CopyFile(LPSTR Source, LPSTR Dest)
Copies a source file to a destination file.

Parameters:
Source: Filename of fie to be copied.

Dest: Filename to be created and copied to.

Returns:
LZH_OK if no errors or error code listed above.

Notes:
The source file is NOT deleted.

DWORD FAR PASCAL GetFreeDiskSpace(LPSTR lpDrv)
Finds the amount of free disk space in bytes.

Parameters:
lpDrv: NULL terminated string containing drive letter to check.

Returns:
0 if an error occurred or bytes free on given disk.

Notes:
Only the first character in the string is actually worked on. The rest
of the string is ignored. To obtain the size of the default (current)
disk, make the first character ASCII 64.

---

int FAR PASCAL SetFileStatus(LPSTR Filename, int Status)
Sets status of file.

Parameters:
Filename: Name of file to change the status of.
Status: Attribute bits. Use LZH constants listed above.

Returns:
LZH_OK if no errors or error code listed above.

Notes:
Attribute constants are LZH_READONLY, LZH_HIDDEN and LZH_SYSTEM.

Bubble Help DLL

Bubble Help window system for Visual Basic
Version 0.1 1/1/92 by Mark Gamber

A feature I liked in C but found unacceptable slow in Visual Basic was a small window available for short text messages such as on the spot help or information. So I concocted this "Bubble Help" system for use from Visual Basic which pretty much does the same thing I do in C. What it amounts to is two functions from Visual Basic and a method of calling the functions. Allow me to expand a bit on this...

In order to display and delete the windows, you need of method of doing so. The example provided uses the top level form and the MouseDown routine. If the mouse button is number 2, the right button, a global variable, wnd, is tested see if it's zero. If so, a call is made to CreateBubble() which returns the handle of the window created. The handle is then stored in wnd so there aren't a zillion windows all over the place. When MouseDown is called again with the button equal to 2, it again tests wnd for zero and, since it's not, calls DeleteBubble() to delete the window created before. This is the method I used and it works well. It works on any object that can support MouseDown routines, such as a picture button. You can use just
about anything you'd like, however. Perhaps a key combination or double click from the mouse, for example.

Since this is fairly easy to do from C, I aimed this primarily at Visual Basic. To use the DLL from Visual Basic, you need to do the following:

In the Global module, enter these lines:

```
Declare Function CreateBubble% Lib "bubble.dll" ( ByVal x%, ByVal y%,
                                                  ByVal xs%, ByVal ys%,
                                                  ByVal title$, ByVal txt$ )
Declare Function DeleteBubble% Lib "bubble.dll" ( ByVal wnd% )
```

Remember, the lines cannot be split as they are above. It's only for clarity. Once done, you may call the functions in this manner:

wnd% = CreateBubble(xpos%, ypos%, xsize%, ysize%, title$, text$)

Variable wnd% is an integer type and is the handle of the window created. If an error occurred, the return value is ZERO. You MUST retain this value (if not ZERO) in order to later delete the window. Xpos and ypos set the window position and xsize and ysize set the size of the window. To prevent surprises, set the Visual Basic form ScaleMode to Pixels since that's what Windows uses to position the help window. Title$ is a string which is displayed in the caption of the
help window and text$ is the actual text which appears in the window. The text$ variable (or literal string) may be up to 256 characters and the same goes with the title, although that would look mighty ugly.

When the text is displayed, it is automatically broken to fit within the size provided and will expand tabs embedded in the string. In addition, it is auto--matically centered so all you need to provide is the raw text to display. If you are creating a relatively short window width-wise, be sure the window is high enough to display all the lines if the text is broken.

Enough of that...it's time to play, I think. The best way to figure out how to use it is to try it. Hopefully, this will make someone (besides me) happy.

As always, suggestions, complaints and general praise may be directed to me via America Online, mail address MarkG85. And, as always, this is completely free as long as you agree not to hold me liable for anything you think may have caused anything. That is, use at your own risk.

MHelp VBx

READ.ME File for MHELP.VBX

This DLL is provided as a courtesy to the Visual Basic programming community by MicroHelp, Inc. MicroHelp is the leading publisher of add-on products for Microsoft Visual Basic, Microsoft QuickBASIC and Microsoft BASIC Professional Development System.

The routines in the accompanying DLL represent only a handful of the routines that are in MicroHelp Muscle. Muscle includes over 400 assembly language and other routines that add speed and functionality to Visual Basic programs. For more information on MicroHelp's products, call 1-800-922-3383 or (404) 594-1185. You can also write to:

MicroHelp, Inc.
4636 Huntridge Drive
Roswell GA 30075-2012
USA

FAX: (404) 594-9629

The files that are included as a part of this package are:

READ.ME	This file
MHELP.BI	Text file containing routine declarations
MHELP.FRM	Form used with MHELP.MAK
MHELP.MAK	Project file demonstrating the routines
MHELP.VBX	The DLL itself

USING THE DLL

MHELP.BI

In order to use the DLL, the file MHELP.VBX must be in your PATH.

In addition, the routines you wish to use must be declared for VB. The easiest way to accomplish this is to insert the contents of MHELP.BI into your global module.

Once the routines have been declared in your app, using them is simply a matter of invoking them. Please see the example application MHELP.MAK for real code examples.

ROUTINE DESCRIPTIONS

MhCtrlHwnd% returns the HWND of any control. This value is useful for passing to Windows API routines. To use the routine, simply pass the CtlName of the property. For example:

```
' Assuming you have a text box control named "Text1"
Text1Hwnd% = MhCtrlHwnd(Text1)
```

The function result Text1Hwnd% can be used for any Windows API routine that requires an "hwnd".

MhPeekByte is similar to QuickBASIC's PEEK statement. Instead of using DEF SEG, you pass a corresponding value in Segm%. For example, suppose in QuickBASIC you have the following:

```
DEF SEG = &H40
X% = PEEK(&H10)
DEF SEG
```

To read the same value using MhPeekByte:

```
X% = MhPeekByte%(&H40, &H10)
```

MhPokeByte is similar to MhPeekByte, but is used to write a value. For example, suppose you have this code in QuickBASIC:

```
DEF SEG = &H40
POKE &H10, 65
DEF SEG
```

The equivalent code for MhPokeByte is:

```
MhPokeByte 65, &H40, &H10
```

The balance of the routines are used *exactly* like their PDS 7.x counterparts:

Inp%, Out, VarPtr%, VarSeg%, VarSegPtr%, SAdd%, SSeg%, SSegAdd&

Note that for variable-length strings, you should use Sadd%, SSeg% and SSegAdd&. The Varxxx routines can be used on all other types of variables.

All routines are provided on an "as-is" basis, without warranty of any kind. If you have a question about using the routines, please leave a message in the MSBASIC conference of CompuServe, in subtopic #5 (Visual Basic). Please do not call MicroHelp for support for this free DLL.

HotKey DLL

TSR Type Processing in VB
by Jonathan Zuck

HOTKEY.DLL is designed to provide hotkey support to Visual Basic as this is not directly supported in VB. You can define multiple hot keys in your application and the DLL can be used by multiple applications simultaneously.

The Point:

The point is that you might want to create a TSR type utility in VB that "pops up" or performs some background task given a certain "hot key." For example, let's say that you want to be able to insert the current date the cursor location of whatever text editor you are using. This is normally impossible to accomplish in VB. Not anymore! Now you can create a utility that has a hot key of Ctrl-D that, while as an icon, inserts today's date in whatever application you are using. Another example might be a popup calendar application. Normally, the user would have to double-click on your icon, but now you can define a key that brings your application to the fore front for immediate use. I am certain, that when you think about it, you will be able to come up with much more interesting ideas, but those are just a couple.

Well, HOTKEY.DLL has two functions: CreateHK and KillHK. The former establishes a hotkey and the latter gets rid of them. The syntax for them is as follows:

hHotKey = CreateHK (VKCode, Shift, Wnd, UserVK)

VKCode is an valid virtual key code. These are the same codes used in the KeyDown and KeyUp event handlers. The values for the different keys can be found in CONSTANT.TXT with the KEY_ prefix.

Shift is the mask for the shift keys you want included in your hot key. These are the same as those used in the KeyDown and KeyUp events in the "Shift" parameter. These values can be added together for multiple shift keys. This concept is explained on page 277 of the Programmer's Guide.

Wnd is the hWnd property of the Form that you want to receive "hot key events." Therefore, you would normally simply pass hWnd as this parameter.

UserVK is the value you want sent to your window when your hot key comes up. This can be any value you want it to be. It is best to start with values over 144 as this is where the normal Key_Codes end. This can be *any* valid integer

hHotKey is returned by the function as the "handle" to the hotkey. This is used only to "kill" the hotkey later on. This will be set to (-1) if the hot key you selected is already in use. It will return a (-2) if there are no more hot keys. You are limited to 1024 simultaneous hot keys on your desktop. Note: this does not include, any

hot keys that might be defined in the Windows Macro Recorder. Those are independent of HOTKEY.DLL. Also, HOTKEY.DLL *cannot* detect if a hot key is already taken by the Windows Macro Recorder. Be careful of overlap!

Once you have created as many hot keys as you want, you need to create a KeyDown event handler for that form. This event is rarely used because normally, it requires that the form have focus. This can only happen when there are no controls on the form or they are all disabled. Never fear, however, HOTKEY.DLL can easily coexist with normal KeyDown processing as long as you define UserVK codes that to not represent valid virtual key codes. Anyway, in the KeyDown handler, you will see two parameters: KeyCode and Shift. The only one of interest to us is KeyCode because the event will only be triggered if all the proper shift keys are pressed! What the KeyCode parameter will contain is the UserVK code that you specified when creating your hot key. You can then use SELECT CASE or something to process the different hot keys that you have created. All this can transpire with your app as an icon and with you app never receiving focus!

When Unloading Your Form:

When you are unloading your form, please call KillHK and pass the "handle" returned to you by CreatHK. Not only will this allow other applications to use those hot keys but HOTKEY.DLL does a lot of cleanup when a hot key is "killed" so that key processing is as fast as possible. When you consider that the DLL has to filter *every* keystroke, it is imperative to processes keys as fast as possible. I think it's pretty quick but I would like to hear comments from you on any performance hits you feel you are taking. I don't think the human eye will be able to notice.

The Example:

HOTKEY.EXE (HOTKEY.* source files) is a *very* simplistic demo of some of the features of HOTKEY.DLL. Essentially, it loads itself as an icon and then loads NotePad. You may type freely in notpad but when you use the following key combinations, funning things happen, all with NotePad never losing focus:

F7	HOTKEY.EXE exits
Shift-F9	The Icon toggles between two pictures
Ctrl-T	The Current time is inserted into Notepad
Ctrl-D	The Current date is inserted into Notepad
Shift-Ctrl-Alt-F10	A message appears

How did I do it? (For those who care...)

HOTKEY was written in Turbo Pascal for Windows. I like Pascal better than C and, as a matter of priciple, I would choose to use the tool that cost me $199 and included a compiler, windows based debugger, the Whitewater Resource Toolkit all in a Windows based development environment (hey, I thought it was MS who was hot on Windows!) that allows me to create *one* source file instead of four (with all sorts of crazy link options and import libraries!). The C/SDK combo cost me close to $1,000 and it is a PITA to use.

HOTKEY was not implemented as a custom control because: 1) it would have been more trouble than it was worth. 2)it probably would have required multiple keyboard hooks instead of one

which would degrade performance. 3)it would only work with VB and we use lots of other similar tools and finally 4)it would have to have been done in C! Argh! Besides, Crescents is creating a low-level keyboard handler control that I will simply buy.

Essentially, HOTKEY has four routines. There is an itialization routine that dims an array of record variables (you know TYPE..END TYPE) to 1024 elements. It then sets some global variables. Finally, it makes a call out to SetWindowsHook, telling it to trap all keys and to filter them through my keyboard handler: KBProc

CreateHK searches through the array to see if the the hot key is already taken. If not, it inserts the passed parameters into the next possible element of the record variable array and increments the count of hotkeys in the system. The long integer returned by CreateHK is actually a combination of your hWnd and your UserVK. This way, the hotkey is uniquely identified.

KillHK moves everything, below the hot key being removed, up one element in the record variable array and decrements the total. I chose to take the extra time to do this here rather than make the search longer for every keystroke.

KBProc constructs a mask using GetKeyState for the SHIFT and Ctrl keys and using lParam for the ALT key. It then scans through the record variable array looking for the same keycode and mask. If it finds it, it passes the WM_KEYDOWN message to the corresponding hWhd, passing the UserVK that you provided as the parmeter. If they key is not found, it is passed on to the next keyboard handler which is usually the application you are working in.

The exit procedure, unhooks KBProc from the keyboard chain and frees up the memory taken by the record variable array. That's about it. I hope you are able to find this useful in your applications and I welcome any suggestions or comments.

Jonathan Zuck
User Friendly, Inc.
07-26-1991 (I just hit Ctrl-D!)

Program Hotkey's

I've included two Hotkey's for the most used functions of VBToolbox to increase usability and convenience.

Shift+F9: This is the hotkey for the Save & Run procedure. It can be used in place of F5 or Shift+F5. This hotkey will save all project files before entering run mode protecting you from lost work and frustration if the system goes down. Both F5 and Shift+F5 remain functional therefore if you wish to test a piece of code without saving, both options are available to you.

F6: This hotkey automatically prints any text you have selected within a form. Great for examining code while debugging. Select the text you wish to print with the mouse, and press F6, **It's that simple.**

Sending Escape Sequences

SENDING ESCAPE SEQUENCES FROM VISUAL BASIC TO HP III PRINTERS

==========

Author: Peter O'Rourke (100064,475)
Company: Tunbridge Wells Equitable Friendly Society
Date: 9th May 1992

PURPOSE

This document discusses a technique for sending printer ESCape sequences from Visual Basic to an HPIII, something which a number of people have been experiencing problems with. The text might be of interest to users of other printers if they too are having problems.

INTRODUCTION

I have been working on a Visual Basic project for some time now. One of the functions that the system needed to perform was the sending of ESCape sequences to our HPIII printers. I was very disappointed to discover that it was not possible to do so.

Despite extensive conversations with Microsoft UK & USA we were unable to come up with a solution. We tried using the Windows API function PASSTHROUGH without any success. We tried printing direct to LPTx also without success. Either of these solutions would probably

work providing you sent ALL printer output via the same method. i.e You must NOT try to intersperse Visual Basic's internal printing commands. The major downside to this approach of course is that you lose all of VB's printer handling functions. e.g. Font selection, size,

attributes and text alignment features.

SOLUTION

Robert Enigigl (76701,155) @ Microsoft suggested that I try using using the API function TextOut, but only if I were prepared to send ALL of my output via this method. I knocked together a test program, which didn't work either. Then I discovered, quite by accident, that if I printed something via Printer.Print first, and then sent my ESCapes via TextOut,

suddenly everything worked!

Now this didn't quite seem correct, since the function I had written to use TextOut actualy sent something via Printer.Print before it called the function, so that I could get the Printer Handle (HDc). After further investigation I discovered that it was not WHAT I was printing

before I sent my ESCape sequences, but WHICH FONT I was using that determined whether or not the ESCape seqences would work.

I then changed my code so that I was sending the ESCapes via the Printer.Print ESCString$ method, which of course is where I started some time ago now, and discovered that this worked fine as well.

As with most things in life the final solution is so simple and obvious, that you want to kick yourself for not having thought of it in the first place. Make sure you are using the correct font BEFORE you send your ESCape seqences. Here is my sample code, which you can modify

to suit:-

```
Printer.Fontname = Printer.Fonts(ANumber)
```

```
Printer.Print Chr$(27) + "&f1001y3X"
Printer.Print Chr$(27) + "&f1004y3X"
Printer.Print Chr$(27) + "&f1010y3X"
Printer.Print Chr$(27) + "&f1024y3X"
Printer.Print Chr$(27) + "&f1042y3X"

Printer.Print "The rest of my data."
```

N.B. The above ESCape codes instruct an HPIII to invoke five prestored macros which go to make up a composite background form for my app.

By default, my setup will cause Visual Basic to use Courier as it's initial printer font. If I send the above ESCape sequences with Courier as my default font, the ESCape character (1Bh) is changed to 7Fh, which of course means nothing to the printer. This causes the following nine bytes in the ESCape sequence to be interpreted as text.

The following table list the fonts that VB can access in my environment, and it shows whether or not the ESCape sequence is sent intact, or modified in some way. My environment consists of Windows 3.1 configured to use the HPIII printer driver, which VB reports as having 16 supported fonts.

Font No.	Font Name	Does It Work?
1	Courier New	Yes.
2	Times New Roman	Yes.
3	Wingdings	Yes.
4	Symbol	Yes.
5	CG Times(WN)	Yes. (Tested to printer correctly)
6	Courier	No. Escape (1Bh) becomes 7Fh.
7	Fences	No. Escape (1Bh) becomes 95h.
8	Line Printer	No. Escape (1Bh) becomes 7Fh.
9	MT Extra	No. Escape (1Bh) becomes 95h.
10	MT Symbol	No. Escape (1Bh) becomes 95h.
11	Symbol	Yes.
12	Univers(WN)	Yes.
13	MS Line Draw	No. Entire ESCape sequence lost!
14	Roman	No. Entire ESCape sequence lost!
15	Script	No. Entire ESCape sequence lost!
16	Modern	No. Entire ESCape sequence lost!

N.B. The only font that I have actualy printed via, and therefore know will work, is number 5 CG Times(WN). I have only examined the disk file created when using the other fonts so I cannot guarantee that there won't be somthing else in the output that might cause problems.

THANKS

My thanks to the various people on Compuserve who offered advice along the way, but in particular Robert Einigl (76701,155) and Rick J. Andrews (70742,1226). Live long and prosper.

Regards

Peter O'Rourke (100064,475)

MHELP.BI

```
' Declarations file for mhelp.vbx
'
'   A routine's declaration must included in any VB program which uses
'   a routine from the MHELP.VBX DLL.
'--------------------------------------------------------------------
Declare Function MhCtrlHwnd% Lib "mhelp.vbx" (A As Control)
Declare Function MhPeekByte% Lib "mhelp.vbx" (ByVal Segm%, ByVal Ofst%)
Declare Sub MhPokeByte Lib "mhelp.vbx" (ByVal ByteVal%, ByVal Segm%, ByVal
Ofst%)
Declare Function Inp% Lib "mhelp.vbx" (ByVal PortNum%)
Declare Sub Out Lib "mhelp.vbx" (ByVal PortNum%, ByVal DataByte%)
Declare Function VarPtr% Lib "mhelp.vbx" (Varbl As Any)
Declare Function VarSeg% Lib "mhelp.vbx" (Varbl As Any)
Declare Function VarSegPtr& Lib "mhelp.vbx" (Varbl As Any)
Declare Function SAdd% Lib "mhelp.vbx" (Lin$)
Declare Function SSeg% Lib "mhelp.vbx" (Lin$)
Declare Function SSegAdd& Lib "mhelp.vbx" (Lin$)
```

Grid VBx

"Explanation of the Grid Control"
by Jonathan Zuck
User Friendly, Inc.

What I have tried to do here is to document the various properties of the Grid control (including a couple I added), for those who do not read C. I am going on the assumption that the properties that appear in the VB editor are self-evident. Be sure to let me know if that is a flawed assumption. Accordingly, I will be documenting the run-time properties only.

The following two properties can be retrieved or set to determine or set the cell in the grid that currently owns focus. You must change these settings in order for the TEXT, ROWHEIGHT, COLWIDTH, COLALIGNMENT properties to have meaning. Each of these operates on the cell that currently owns focus.

ROW (Int)
The Row of the Cell that has focus
This can be both retrieved and set at run-time.

COL (Int)
The Column of the Cell that has focus
This can be both retrieved and set at run-time.

TEXT (String)
The contents of the current cell.
This can be both retrieved and set at run-time.

ROWHEIGHT (LONG)
The hight of the row containing the current cell
This can be both retrieved and set at run-time.

COLWIDTH (LONG)
The width of the row containing the current cell
This can be both retrieved and set at run-time.

COLALIGNMENT (Int)
The column alignment of the column of the current cell
This can be both retrieved and set at run-time.

CELLSELECTED (BOOL)
Returns TRUE/FALSE if any cells are/are not currently selected.
Read-only at run-time

SELSTARTROW (Int)
The starting row of the selection block
This can be both retrieved and set at run-time.

SELENDROW (Int)

The ending row of the selection block
This can be both retrieved and set at run-time.

SELSTARTCOL (Int)

The starting column of the selection block
This can be both retrieved and set at run-time.

SELENDCOL (Int)

The ending column of the selection block
This can be both retrieved and set at run-time.

CLIP (String)

The currently select text (tab and CR/LF delimited)
This can be both retrieved and set at run-time.
Note: This property can be used to retrieve and save delimited text files or perform advise links with EXCEL.

TOPROW (Int) (I added this)

The Top most visible row in the grid.
This can be both retrieved and set at run-time.

LEFTCOL (Int) (I added this)

The Left most visible column in the grid
This can be both retrieved and set at run-time.

Note: I added the above two properties because there was no way to programatically scroll the grid. Using these properties, you can force a particular cell (or just a row or column) to be the visible origin of the grid. The top row and left column are contained by the number of fixed rows and columns and the number of remaining rows and columns. Therefore, you cannot use this to set new titles or to bring the last row/column to the origin.

Hope this helps!
JZ

3D Frames DLL

FRAMES.DLL was be presented in my July 1992 WinTechnician column in the Windows Tech Journal. The purpose of the DLL is to demonstrate an easy method of 3D shading as well as one potential application of a technique called subclassing. The DLL works by allowing the developer to "register" a window, specifying the type of frame (0 - out, 1 - in) and the DLL will replace the standard border of the window with a 3D frame whenever it needs painting. It uses subclassing which essentially means that I see messages sent to the window before it does and I act on the WM_PAINT message. This should *not* be confused with subclassing the in the VB CDK sense. This DLL has no VB specific code.

Function Declarations:

Declare Function ControlhWnd Lib "CTLHWND.DLL" (Ctl As Control)

Declare Sub FrameWnd Lib "FRAMES.DLL" (ByVal hWnd, ByVal Style)
Declare Sub UnFrameWnd Lib "FRAMES.DLL" (ByVal hWnd)
Declare Sub ChangeFrame Lib "FRAMES.DLL" (ByVal hWnd, ByVal Style)
Declare Sub FrameChildren Lib "FRAMES.DLL" (ByVal hWnd, ByVal Style)
Declare Sub UnFrameChildren Lib "FRAMES.DLL" (ByVal hWnd)
Declare Sub ChangeChildren Lib "FRAMES.DLL" (ByVal hWnd, ByVal Style)

The functions, exported by the DLL are the following:

FrameWnd (hWnd, Style) - Frame a particular window
UnFrameWnd (hWnd) - UnFrame a particular window
ChangeFrame (hWnd, Style) - Change the style of the frame of a particular window

The following functions behave the same as the ones above but operate on all of the children of the specified window, allowing you to establish framing for an entire form, or frame control with one call.

FrameChildren (hParent, Style)
UnFrameChildren (hParent)
ChangeChildren (hParent, Style)

The reason that there are both individual and "group" functions is to give you the greatest flexibility. For example, you might want to FrameChildren on the whole form, but then UnFrameWnd each command button. Another example, might be that you would want to change the style of a particular window or eliminate shading within a group box. The demo FRAMES.MAK shows each of these commands in action.

You may use any technique you like to retrieve the hWnd of a particular control. The method I have used is to use the DLL I uploaded to CIS a million years ago: CTLHWND.DLL. You can see how this is used from the sample. In any case, you will want to copy the two DLLs into your system directory.

Theoretically, there is no need to call UnFrameChildren (or UnFrameWnd), at the end of your application. This is because I trap the WM_DESTROY message to each of the framed controls and clean them up at that time.

Source Code:

The TPW source for FRAMES.DLL can be found in Lib 9 of CLMFORUM. The source with explanation can befound in the July '92 issue of the Windoes Tech Journal.

For more information or subscriptions to the Windows Tech Journal, call (800) 234 - 0386 or outside of the US, (503) 747 - 0800. The fax number is (503) 747 - 0071. Alternatively, you may reach the editor, J.D. Hildebrand (until he becomes too popular) here on CIS at id: 76701,32. The journal can be reached by mail at:

Windows Tech Journal
P.O. Box 70167
Eugene, OR 97401-0110

Thanks a lot for your interest! -=- Jonathan

Drag & Drop DLL

D&DSERVE.DLL
DLL to Support Win 3.1 Drag 'n Drop Server Capability
by Jonathan Zuck
Published in the August 1992 Windows Tech Journal
This file contains the description for use by VB programmers

Function Summaries:

DropSelItems	Drops the selected files in a multi-select List Box
DropAllItems	Drops all the files in a list box
DropBuff	Drops files contained in a passed buffer

Function Declarations:

```
Declare Function DropSelItems Lib "D&DSERVE.DLL" (ByVal hList, ByVal Path$, ByVal nButton)
Declare Function DropAllItems Lib "D&DSERVE.DLL" (ByVal hList, ByVal nButton)
Declare Function DropBuff Lib "D&DSERVE.DLL" (ByVal Buff$, ByVal nButton)
```

Function Details:

DropSelItems: (D&DTEST1.MAK)

You must have a multi-select list box to use this function. Most of you will be using those list boxes available from third parties such as Crescent, MicroHelp and Sheridan. However, to make the demo more generalized, I modified Costas' MultiPik demo to accomodate the needs of a file list. Unfortunately, this means that it's a little more complicated than it needs to be. The real demo code for this function would look something like this:

```
Sub MyFileList_MouseDown (Button As Integer, X As Single, Y as Single)
        Path$ = Dir1.Path + "\"
        hTarget = DropSelItems (MyFileList.hWnd, Path$, Button)
End Sub
```

where hTarget will be the hWnd of the Window on which the selected files were dropped or Zero if the mouse was not released over a valid window. One caveat is that the left mouse button will generally remove the selections from an extended selection list box (the File manage does a ton of special processing). Therefore, you might want to specify for the user that the Right mouse button be used to drag from the list box.

DropAllItems: (D&DTEST2.MAK)

This function has two advantages. First, it does not require a multi-select list box so those of you without add-ons will be able to use it. Second, it is capabile of dealing with files from multiple directories which DropSelItems (or even FileManager) is not capable of. Keep in mind that since you do not pass the path to this funtion, each file name in the list must be fully qualified. When used, this function will drop all of the items in the list on the target window.

DropBuff: (D&DTEST3.MAK)

This function is provided primarily for those programming systems that are not able to use a standard list box or have special needs. In this case, you are essentially constructing the buffer that will be sent to the target window. The format of this buffer is a fully qualified filename, followed by a null, followed by another filename+NULL, etc., and finally terminated by another null. Visual Basic will add the terminating null so you don't need to worry about the second one. You just need to construct a strong like so:

```
N$ = Chr$(0)
Buff$ = "C:\AUTOEXEC.BAT" + N$  + "C:\DOS\README.TXT" + N$
```

and then pass it ByVal to the function. I dont't generally recommend this function as it is the least intuitive to the user but it might be helpful under special circumstances.

Testing your Code:

I have found WRITE to be the best place to test these functions as it is one of the only WINAPPs that supports the dropping of multiple files (into the client area, *not* on the icon!).

Source Code:

The TPW source code for D&DSERVE is available in Lib 9 of CLMFORUM and was first published in the August 1992 issue of the Windows Tech Journal. For explanations of how these functions work, please contact Oakley Publishing for back issues.

For more information or subscriptions to the Windows Tech Journal, call (800) 234 - 0386 or outside of the US, (503) 747 - 0800. The fax number is (503) 747 - 0071. Alternatively, you may reach the editor, J.D. Hildebrand (until he becomes too popular) here on CIS at id: 76701,32. The journal can be reached by mail at:

> Windows Tech Journal
> P.O. Box 70167
> Eugene, OR 97401-0110

Thanks a lot for your interest! -=- Jonathan

Installation

DISCLAIMER:

VBToolBox is sold "as is" and without warranties as to performance of merchant ability or any other warranties whether expressed or implied. Because of the various hardware and software environments into which VBToolBox may be put, no warranty of fitness for a particular purpose is offered.

Installation:

VB-TB.HLP	This File
VBTBOX2.EXE	VBToolBox executable
TOOLBOX2.CFG	Configuration file
THREED.VBX	System file
SS3D2.VBX	System file
HOTKEY.DLL	System file
_README.TXT	Readme 1st
ADDONS.LST	List of available freeware, shareware VBx, DLL's etc
VENDOR.DOC	Vendor information

Copy the files THREED.VBX, SS3D2.VBX, VB-TB.HLP and HOTKEY.DLL into your C:\Windows\System directory. Copy the files VBTBOX2.EXE and TOOLBOX2.CFG to your Visual Basic directory. VBTBOX2.EXE and TOOLBOX2.CFG **must** be in the same directory. If the file VBTBOX2.EXE is in your VB directory it will call Visual Basic upon start up. Since VBToolbox directly manipulates Visual Basic the toolbox rarely has the focus and F1 will not work with the VB-TB.HLP file. I have set up a default configuration for this file in the call help section of Toolbox2.cfg. If you place this file in a directory other than the default C:\Windows\System make sure to update the file.

Note for Sheridan 3D Widget users:

The file SS3D2.vbx is a runtime only version. If you have your own copy of 3DWidgets use your own file

VB-Comm.VBx

Communications Control for Visual Basic
by Mark Gamber March 1992

This control provides access to the communications ports through the Windows API. To use the control, copy VBCOMM.VBX to your Visual Basic directory. Select the file from the listbox displayed by selecting "Add File..." in the "File" menu. A "port" button should appear in the toolbox. Selecting a port and placing it on a form will display another small "port" button. This is only displayed during design mode and should be treated like a timer control.

Port settings:
These settings define the port setup. They may be set at design time or run time to any value desired within the ranges described:

Comport: 0 = COM1:, 1 = COM2:, 2 = COM3:, 3 = COM4:

Baud: 0 = 300, 1 = 1200, 2 = 2400, 3 = 9600, 4 = 19200

Parity: 0 = Parity Off, 1 = Parity Even, 2 = Parity Odd

Data: 0 = 6 bits, 1 = 7 bits, 2 = 8 bits

Stop: 0 = 1 bit, 1 = 2 bits

Port initialization:
Before using the port, it must be enabled. Use "Comm1.Enable = 1" to enable the port and "Comm1.Enable = 0" to disable (close) the port. Attempts to enable a port already enabled results in error #10001. Bad or unsupported port setup parameters result in either error #10001 or #10002. If an attempt is made to disable an unenabled port, error #10004 will occur. All these errors may be trapped by the Visual Basic program.

Data I/O:
Sending data involves sending integer values of ASCII codes one at a time using "DataChr" or by strings using "DataStr". An example of each:

Single character:

```
Comm1.DataChr = asc( "A" )              ' Send letter "A" out the port
```

String:

```
Comm1.DataStr = "ATZ" + chr$( 13 )      ' Send "ATZ" command to modem
```

Receiving data can be a bit trickier. Internally, a timer is used to poll the port for any characters and, if

one or more are received, it calls the "InQueue" event. The Visual Basic programmer may use the event to read the data into a string or character by character. As long as there is data to be read from the port input buffer, this event is called every time the timer goes off.

An example of "InQueue" code might be:

"InQueue as Integer"

```
if InQueue Then                    ' If not a false firing...
    a$ = Comm1.DataStr             ' Read data into string
    if len(a$) Then                ' If something was read...
        Picture1,Print a$          ' Do something useful
        while len( a$ )
            a$ = Comm1.DataStr     ' While data is being read...
            if len( a$ ) then
                Picture1.Print a$  ' Be more useful
            end if
        wend
    end if
end if
```

End of "InQueue"

Following the code, InQueue is the number of characters waiting to be read. In some chance the routine is called with nothing waiting, this should be checked. Next, we read the port buffer and again check to be sure something was actually read. If so, do something useful and, as long as data is waiting to be read from the port, repeat the process. If using very high speeds on relatively slow machines, the "While" section may cause the program to appear to lock up since there's always data to receive. Omitting that section is less efficient but doesn't attempt to clear the port before exiting. Data may be received in two ways. Incoming data may be read character by character using "DataChr" and as a string using "DataStr". An example of each:

Single character:

```
a% = Comm1.DataChr                 ' Read character from port
if a% <> -1 then                   ' -1 means nothing was there
    do something useful            ' Otherwise, valid ASCII code
end if
```

String:

```
a$ = Comm1.DataStr                 ' Read string into a$
if len( a$ ) then                  ' If something was actually read...
    do something fun               ' See code
endif
```

Attempting to read or write an unenabled port results in an error 10004.

This software is free for all to use given two conditions:

1. Ownership is retained by Mark Gamber as of March, 1992
   (Give credit where credit is due)

2. I'm not liable.
   (What? Me Worry?)

Bugs, suggestions and whatnot may be directed to PCA MarkG on America Online or through Internet to pcamarkg@aol.com, E-Mail only. No other messages or mail may be responded to.

Addons DLL Declarations

```
'
'   Declares for Huge Array Support
'
Declare Function HugeCurrency Lib "vbaddons.dll" (ByVal hArray%, ByVal
                                                  el&) As Currency
Declare Function HugeDim Lib "vbaddons.dll" (ByVal recsize%, ByVal
                                                  limit&) As Integer
Declare Function HugeDouble Lib "vbaddons.dll" (ByVal hArray%, ByVal
                                                  el&) As Double
Declare Function HugeErase Lib "vbaddons.dll" (ByVal hArray%) As Integer
Declare Function HugeGetElement Lib "vbaddons.dll" (ByVal Index%, ByVal
                                        el&, buffer As Any) As Integer
Declare Function HugeInt Lib "vbaddons.dll" (ByVal hArray%, ByVal el&)
                                                   As Integer
Declare Function HugeLong Lib "vbaddons.dll" (ByVal hArray%, ByVal el&)
                                                   As Long
Declare Function HugeNumArrays Lib "vbaddons.dll" () As Integer
Declare Function HugeRedim Lib "vbaddons.dll" (ByVal hArray%, ByVal
                                                  limit&) As Integer
Declare Function HugeSetElement Lib "vbaddons.dll" (ByVal Index%, ByVal
                                    el&, buffer As Any) As Integer
Declare Function HugeSingle Lib "vbaddons.dll" (ByVal hArray%, ByVal
                                                  el&) As Single
Declare Function HugeUbound Lib "vbaddons.dll" (ByVal hArray%) As Long
'
'   Data type for DiskGetFirstFile & DiskGetNextFile
'
Type FindDataType
  reserved As String * 21
  FileAttr As String * 1
  FileTime As Integer
  FileDate As Integer
  FileSize As Long
  FileName As String * 13
End Type
'
'   Bitmasks for FileAttr field of FindDataType
'
Global Const FILE_NORMAL = 0  ' Normal file - No read/write restrictions
Global Const FILE_RDONLY = 1    ' Read only file
Global Const FILE_HIDDEN = 2    ' Hidden file
Global Const FILE_SYSTEM = 4    ' System file
Global Const FILE_VOLID = 8     ' Volume ID file
Global Const FILE_SUBDIR = 16   ' Subdirectory
Global Const FILE_ARCH = 32     ' Archive flag
Global Const FILE_ALLFILES = 63 ' All files


Declare Function DiskGetFirstFile Lib "vbaddons.dll" (ByVal StartString As
String, ByVal AttrFlags As Integer, FindData As FindDataType) As
                                                   Integer
Declare Function DiskGetFreeSpace Lib "vbaddons.dll" (ByVal DriveLetter
                                                  As String) As Long
Declare Function DiskGetNextFile Lib "vbaddons.dll" (FindData As
                                        FindDataType) As Integer
```

```
Declare Function DiskSetLabel Lib "vbaddons.dll" (ByVal DriveLetter As
                        String, ByVal NewLabel As String) As Integer
```