

Visual Basic ToolBox Pro 2.0b

copyright (c) Hal Jurcik 1992

Thanks for trying VBToolBox Pro (2). This is a fully functional Shareware version of VBToolBox for your evaluation. There are only two registration reminders, a nagware start up screen, and monitoring of VB is not integrated into the shareware version. Therefore the shareware version will not minimize or exit in conjunction with Visual Basic. It will have to be treated as a separate program. Also the toolbar will not recede during run mode or for help files, you will have to minimize it manually if you desire.

I'm sure you will find this program indispensable once you begin using it in your programming environment. To order a fully integrated registered version check out the Ordering Info section for more information on price and registration.

[Installation](#)

[Ordering Info](#)

[Save & Run](#)

[ToolBar](#)

[Saving Code as text files](#)

[Printing Code](#)

[Call Text Files](#)

[Call Help](#)

[Launch Programs](#)

[Vbx Files](#)

[hWnd & VB Spy](#)

[Program Hotkey's](#)

[Configuration](#)

Help for common Shareware & PD Visual Basic Addons

[DLL / VBx Reference](#)

[Code Examples & Tips](#)

A rectangular button with a black border and a light gray background. The text "S/Run" is centered on the button in a black, sans-serif font.

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 {Control + F5} 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 haven't given your project a name Save & Run will call the Save As dialog box.

In design mode VBToolbox has a function that will save your work at specified intervals. See the configuration section for more information on configuring the Auto Save feature.

[Configuration](#)



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 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 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 dialog select the Launch option button to access the option list box. Add the title and location of programs you wish to access under each heading. Feel free to add as many as you wish, no limit. If you wish to re-order the list box, while your in the configuration dialog double click on a selection in the list. This will place that selection in the editing box. If you press "Add" without modification, the selection will be re-entered at the bottom of the list.

[Launch Programs]

Title for list box

Exe with path

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 your 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 configuration dialog.

Vbx Location=C:\Windows\System

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

[Configuration](#)



hWnd & VB Spy

In order to use the Windows API functions 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.

New to Version 2 is the VB Spy dialog. This new feature will give you the handle of any control in any active window along with the classname and caption. It will also supply the same information for that controls owner and parent application. Just place the cursor over any control or form you want information on. Perfect for checking API calls!

Note: For an accurate reading of a controls hWnd the control should be active. [not disabled]

Configuration

New to Version 2 is the Configuration Dialog. There is no longer any need to edit the cfg file manually. The command for calling the configuration dialog is located in the text file list box, at the end of the list. Just double click on the name "Configure VBToolBox". The configuration dialog contains two sections. The upper section consists of a series of text boxes to enter instructions for user options. When you tab or click on a text box basic instructions and defaults will be displayed in the information panel located in the center of the dialog box.

When you are finished with your changes make sure to press the **Re-Config button to write the new information to the configuration file. You must exit VBToolBox and restart for changes to take effect.**

Registration #:

This text box displays the status of your program: Registered or Not Registered.

Printer Type:

Default: PrinterType = HP Compatible

VBToolBox supports four basic printer types: HP Compatible, Dot Matrix, Wide Dot Matrix, and New HP. Enter your printer designation.

For **laser or ink jet printers** that use standard sheet sizes and accept standard HP instructions, use the **HP Compatible** designation. This instruction enables both portrait and landscape printing capabilities.

If you have a **standard dot matrix** printer that uses 9.5" perforated paper, use the **Dot Matrix** designation. This instruction disables landscape printing.

If you have a **wide carriage dot matrix printer**, Use the **Wide Dot Matrix** designation. This instruction also disables landscape printing, but resets the line instruction from portrait values to landscape values to take advantage of the increased paper width.

Finally if you have one of the **new HP level IV printers** that bypasses the print manager for its own proprietary driver use **New HP**. At this time I cannot access this driver to automatically re configure the printer orientation. Therefore with the New HP designation everytime you access the print command the printer setup dialog box for your printer will appear. If you have chosen landscape printing make sure you also select landscape in the printer setup dialog to reset your printer, then press ok and the print process will proceed automatically. I know this is an extra step, as soon as I receive a DLL to access these new functions in the HP driver I'll incorporate them in a new version.

Printing Note:

Most printers that have Win 3.1 drivers should accept the HP Compatible API calls (Famous Last Words). If your printer fails to accept these codes and you wish to try landscape printing Use the New HP designation.

Landscape Printing Note:

Landscape printing requires much more memory on disk for temp files than portrait mode. Code in the form of standard txt files printed in portrait mode will generate average temp file sizes of about 1k per page. For an average project with about 80 to 100 pages of code you might expect your temp files to average about 100k for this project printed in portrait mode. Printed in landscape mode this same project with 80 to 100 pages of code will generate about 8 mgs of temp files to rotate all that text. These temp files are generated in your temp directory not in your permanent swap file!! Make sure you

have enough memory on disk to print large projects in landscape mode. If you don't have enough memory for your temp files windows will not handle the shortage. Generally your system will lock up, and you will have to reboot and loose all your unsaved work. For all those people out there that have ram drives of one or two mgs for their temp files, your asking for major problems. For printing large projects in landscape, 5 to 6 mgs of free disk space is bare minimum. The more you have the faster and easier your printing jobs will proceed with all your Windows programs.

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

Launch at Startup: / Launch State:

VBtoolbox will launch one program upon start up in addition to VB. If you have a program you wish to launch at start up [[example VB Assist](#)] with Visual Basic enter the full path and exe location in text box #3 and the start up state in #4. For startup state: 1 = normal, 2 = minimize.

Button Launch: / Btn Launch State:

The left arrow button in VBToolBox is user configurable. in effect this is a quick launch button. If you have a program you use constantly within VB enter its full path in this text box and the launch state in the Btn Launch State text box. [1 = normal, 2 = minimized]. This function is identical to the launch programs list box. It uses the shell command, therefore anything including switches can be passed. For a program you use often this button is a convenience to save keystrokes.

Current Project:

VBToolBox will also launch your current project on start up. Enter the path and *.mak file location in this text box.

Vbx Location:

Enter the full path of the location of your Vbx files. VBToolBox uses this variable to find individual Vbx files. Default = C:\Windows\System

ToolBar Btn #1; Btn #1 Cmd\$ / Toolbar btn #2; Btn #2 Cmd\$

The left two buttons on the floating toolbar are user configurable; You can set them to use the shell function to launch a program, or the sendkeys function to access menu items. In the text boxes labeled ToolBar Btn #1 and ToolBar Btn #2 enter **SendKeys** to enable the SendKeys function for that button, or **Shell1** to enable the shell function in a normal state, or **Shell2** to enable the shell function in a minimized state.

In the Btn #1 Cmd\$ and Btn #2 Cmd\$ text boxes enter the command string. If your using the shell function there are examples in the VB help file. If you are using the SendKeys function to access a menu item **omit the quotation marks and enter only the keys to be sent. Also use only the Alt + underscored letters in the menu. Function Keys may not be processed properly.**

example File open would be Alt + File + Open; enter %FO in the text box.

Page Width Land; Page Width Port; Max. Lines Port; Max. Lines Land

These four text boxes allow you to customize the settings for your printer. With the addition of true type and all the print cartridges for lasers today a standard default for characters per line and lines per page is something of the past. The default settings should work for most printers. If you are loosing a character or two at the right of the page, or if the last two lines of a page are printed to a new page just adjust the

appropriate value. Conversely if you are using non standard paper, your printer uses a smaller default type face, etc. and you have extra room on the page feel free to experiment to maximize your output per page.

Page Width Land;

This is the # of characters per page your printer is capable of printing in landscape mode.
default = 102

Max. Lines Land;

This is the # of lines per page your printer is capable of printing in landscape mode
default = 43

Page Width Port;

This is the # of characters per page in portrait mode.
default = 77

Max. Lines Port;

This is the # of lines per page your printer is capable of printing in portrait mode
default = 60

Note for Deskjet 500c and 550c color printer users:

This is specifically for 550c owners. I'm assuming that 500c printers have similar characteristics. If you have this printer you already know that to accommodate the color cartridge the page has been shifted up 1/4" and the default bottom margin has been increased to 7/8". The values for this printer are

Page Width Land = 100

Page Width Port = 77

Max Lines Land = 43

Max Lines Port = 55

Minimize on start / Always on top:

These two check boxes control the properties of VBtoolBox on start up. Minimize on start will start VB toolbox in a minimized state. Always on top will keep the toolbox the top window at all times for easy access of its functions. If you minimize the toolbox this property is disabled for the icon.

Tbox Loc:

This selection controls the position of the floating toolbar and ties it to the Visual Basic Main window. Possible values are 0 through 4.

640 x 480 & 800 x 600 in a 14 or 15 in monitor:

default = 4 0, 2 or 3 also available

This places the toolbar next to the Visual Basic toolbar. In 640 resolution this covers the Twips position values at the right. If you need to use this for positioning you can minimize the floating toolbar.

"0" eliminates the toolbar

"2" places the floating toolbar in the white menu section next to the help menu command

"3" places the floating toolbar to the far right under the main window, above the project window.

1024 x 768 on a 17 in monitor;

default = 1 0, 1, 2, 3 or 4 available

This places the floating toolbar next to Visual Basics default toolbar.

"0" eliminates the toolbar

"2" places the floating toolbar in the white menu section next to the help menu command
"3" places the floating toolbar to the far right under the main window, above the project window.

Feel free to experiment with placement for your way of working.

The best feature of the VB floating toolbar is its floating property. It always stays on top no matter what window is active. Especially useful when developing on a 14 or 15 in monitor, you can now maximize your code windows and use the floating toolbar to access menu functions for copy, cut, paste or any other menu item you wish to program in. Give it a try you'll never be able to go back.

AUTO SAVE:

The auto save feature of VBToolBox will save your project at specified intervals. **This function works only in design mode.** If you are debugging die hard that spend hours toggling between break and run, use the Save & Run button or press Ctrl+F5 to save your project before entering run mode.

Valid entries for this text box are: 100 to 999 seconds.
Default = 600 {Save at 10 min intervals}
0 [zero] Disables Auto Save

Launch Programs:

VBToolBox allows you to launch programs from within Visual Basic. Select the launch option button to access the list box information. Under the heading "Title for list box" enter the title you wish to see in the list box to access this program. Under the heading "Exe with complete path" enter the path of the program you wish to launch. If you wish to edit or delete existing entries, double click to make your selection. Once selected you can make changes to the existing values and press the add button, or press the delete button to delete that entry.

For current VBToolBox users there is no need to add commas in this version

"Default Sample"

Title for List box	Exe with complete path
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

Text Files

If you have text files of constants for your Visual Basic add on programs, or programming examples you've collected from bulletin boards, enter their location and titles in this section and you will be able to access them from within VB. Choose the text option button to access the list box.

Title for list box = Program name that will be entered into the VBToolBox list box.
Path of Text Viewer = Program used to view this text file, (Notepad, Write, etc.)
Text File Location = Location of text file.

"Default Sample"

Title for list box	Path of text Viewer	Text file Location
--------------------	---------------------	--------------------

VBpro Setup Kit	C:\windows\notepad	C:\vb\setupkit\readme.txt
VBpro HugeArray	C:\windows\notepad	C:\vb\samples\hugearray\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

Help Files

If you have any addons for Visual Basic one problem is sorting out all the accompanying help files. Generally you must assign them an icon and return to the program manager to access them. VBtoolBox allows you to access these files from within VB. Select the help option and call up the help file list box, enter the title and location of your help files.

Title for list box = Title of help file to be entered into the VBToolBox selection list box.

Help File Location = 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.

"Default Sample"

Title for list box	Help File Location
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

Important:

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

Correcting Errors:

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, and path errors. Check your Title\$ entry, it must be 20 characters or less.

If you have any comments or suggestions for future versions you can reach me on 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 Version 2.

Only \$24.95 & shipping and handling

US and Canadian Orders

Foreign Orders

Reg.	\$24.95	Reg.	\$24.95
S&H	\$ 5.00	S&H	\$ 7.00
Total	\$29.95	Total	\$31.95

If you are paying by check or money order

Make checks payable to:

Hal Jurcik

7819 Vanport Ave.

Whittier, CA. 90606

If you wish to Order using Master card, Visa, or American Express you may call our registration service:

Ask for Product Item # 10735

Public Software Library Registration Service:

800-242-4775

For foreign orders

713-524-6394

Fax# 713-524-6398

Note: These phone numbers are for CREDIT CARD ORDERS ONLY !!

The operators are order takers only. They cannot give any specific answers or support for any program. Any questions about this program, volume discounts, dealer pricing, technical questions or problems of any kind Contact me on COMPUSERVE or at the above address.

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.

Features:

User configurable Floating Toolbar

VBToolBox Instant Watch

Save & Run; Saves you work before entering run mode

Auto Save, Saves your work at specified intervals

Load your current project on start up

Automatically start 2nd program on start up

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

VB Spy program to find the hWnd's of any control, form, or window along with its parent /owner

Launch any program from within Visual Basic

Easy access to all third party help files from within VB

VBToolBox Help file

Ability to automatically save your project as a series of text files

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!**

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

EnumFonts Dll

VB Comm 2.0 Win 3.1

WHelpLib.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](#)

[Sending Escape Sequences](#)

[Text Box routines](#)

[Print Preview Example](#)

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 **Version 1**. 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.

Suggested 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:

Landscape printing requires much more memory on disk for temp files than portrait mode. Code in the form of standard txt files printed in portrait mode will generate average temp file sizes of about 1k per page. For an average project with about 80 to 100 pages of code you might expect your temp files to average about 100k for this project printed in portrait mode. Printed in landscape mode this same project with 80 to 100 pages of code will generate about 8 mgs of temp files to rotate all that text. These temp files are generated in your temp directory not in your permanent swap file!! Make sure you have enough memory on disk to print large projects in landscape mode. If you don't have enough memory for your temp files, windows will not handle the shortage. Generally your system will lock up, and you will have to reboot and loose all your unsaved work. For all those people out there that have ram drives of one or two mgs for their temp files, your asking for major problems. For printing large projects in landscape, 5 to 6 mgs of free disk space is bare minimum. The more you have the faster and easier your printing jobs will proceed with all your Windows programs.

**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.

For information on configuring VBToolbox for your printer type also see:

[Configuring VBToolBox](#)



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 its 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 its 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 category.

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

THE INFORMATION PROVIDED IN THIS SOFTWARE AND ANY DOCUMENTATION MAY ACCOMPANY THIS SOFTWARE IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND/OR FITNESS FOR A PARTICULAR PURPOSE. THE USER ASSUMES THE ENTIRE RISK AS TO THE ACCURACY AND THE USE OF THIS SOFTWARE.

Copyright 1991 by:

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

Windows and Visual Basic are trademarks of Microsoft Corporation

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 in 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 repetitious, 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 receive 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 information, 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 file 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

You are free to use the routines provided in this DLL without restriction. That means you may distribute the DLL with your .EXE files that require the DLL.

You may also distribute the DLL and related files (see below) in archive or ZIP format. In fact, we encourage you to distribute them! All we ask is that you place all the files listed below in the archive/zip, without modification.

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
Copyright 1991 User Friendly, Inc.

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 do 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 your 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 process 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 notepad 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 principle, 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 initialization 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 hWnd, passing the UserVK that you provided as the parameter. 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.

- Ctrl+F5:** 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.**
- Ctrl+F9** This hotkey accepts the defaults for instant watch and sets your watch variables with one action.

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 actually 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 sequences 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 sequences. 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 HP III 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 HP III 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 actually 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 something 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$)
```

3D Frames DLL

FRAMES.DLL was 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 be found in the July '92 issue of the Windows 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:

DropSellItems 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 DropSellItems 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:

DropSellItems: (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 = DropSellItems (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 manager 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 capable of dealing with files from multiple directories which DropSellItems (or even FileManager) is not capable of. Keep in mind that since you do not pass the path to this function, 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 string like so:

```
N$ = Chr$(0)
```

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

and then pass it ByVal to the function. I don'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:

File List:

Program files,	
VB2_TBOX.HLP	This File
VB2_TBOX.EXE	VBToolBox executable
TOOLBOX2.CFG	Configuration file
THREED.VBX	System file
SS3D2.VBX	System file
COMMDLG.VBX	System file
HOTKEY.DLL	System file
Installation program files	
_README.TXT	Readme 1st
SETUP.EXE	Setup initialization
SETUP.LST	Setup initialization file
SETUPKIT.DLL	System file
VBSETUP.EXE	VBToolbox Setup program

To install VBToolBox Pro (2) run the setup program from the program manager or file manager and follow the Online instructions. Installation including icon creation is automatic.

The installation program is set up to work properly with Windows Program Manager and Norton Desktop for Windows. If you have a third party shell that is not recognized by the installation program you can install VBToolbox manually. The files on disk are not compressed, the last letter was renamed with an underscore "_" to prevent the usage of the program files from the floppy during the installation process. Rename each file replacing the underscore with the appropriate letter in the above program file list and follow the manual installation instructions.

MANUAL INSTALLATION:

1. Copy the files THREED.VBX, SS3D2.VBX, COMMDLG.VBX, VB2_TBOX.HLP and HOTKEY.DLL into your C:\Windows\System directory.
2. Copy the files VB2_TBOX.EXE and TOOLBOX2.CFG to your Visual Basic directory.
3. VB2_TBOX.EXE and TOOLBOX2.CFG **must** be in the same directory. Since VBToolbox directly manipulates Visual Basic, if these two files are not in the VB directory some functions will not work properly. VBToolbox rarely has the focus and F1 will not work with the VB2_TBOX.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.

4. Once properly installed, create an icon for VBToolBox in any program manager group and double click on it. Visual Basic will load automatically. To configure VBToolBox for your system click the top left button in the toolbox to call up the text file list box. Near the bottom of the list there will be an entry "Configure VBToolBox". Double click to open the Configuration Dialog Box. As you select a text box information about its function and defaults will be shown in the information panel in the center of the screen. For a more detailed explanation of each function see the Configuration section in this help file

Configuring VBToolBox for your system

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
```

EnumFonts DLL

Enumfonts call from VB

Copyright (C) Telelink Systems 1991, All rights reserved

Phone: (916) 332-2671 Fax: (916) 332-2529 Cserve: 70523,2574

Written by Erez Carmel, Sacramento, California

Description:

The DLL allows Visual Basic to call Windows API function EnumFonts.

The same function is called multiple times, as described here:

1. Call VBEnumFonts to find how many typefaces are there
2. Allocate an array to accept typefaces
3. Call VBEnumFonts to get the typefaces
4. For each typeface, call VBEnumFonts to find how many fonts are available.
Then allocate 3 arrays of the appropriate size, and call VBEnumFonts to get information about the fonts.

The interface is declared by:

Declare Function VBEnumFonts Lib "enumfont.dll" (ByVal hDC%, ByVal lpFaceName As Any, lpLogFontArray As Any, lpTextMetricArray As Any, nFontTypeArray As Any, ByVal nArraySize%) As Integer

Where:

hDC%	Is the context handle to the default printer, typically provided by Printer.hDC
lpFaceName	Is either a null, or the name of the desired typeface. If Null, the function will return one font for each available typeface. Reminder -- Null in VB is indicated by "byval 0&"
lpLogFontArray	Is either a null, or the first element of an array of type LOGFONT. The array should be large enough to accept as many items as specified by the last parameter.
lpTextMetricArray	Is either a null, or the first element of an array of type TEXMETRIC. The array should be large enough to accept As many element as specified by the last parameter.
nFontTypeArray	Is either a null, or the first element of an array of type

integer. The array should be large enough to accept as many elements as specified by the last parameter.

nArraySize%

Is the number of elements expected, e.g. the size of the three arrays. Can be 0, in which case no data will be transferred to the arrays. The function will return data up to the number of elements specified.

Function Returns: The number of available fonts which meet the specification. If lpFaceName is null, this would be the number of typefaces. If lpFaceName specifies a typeface, this will be the number of fonts under that typeface.

You are free to use this code and incorporate it in your programs. If you find it useful, a \$7.50 payment will be appreciated. In return, you will be informed of future changes/enhancements.

The author makes no warranties, express or implied, oral or written, including any implied warranties of merchantability or fitness for a particular use. In no event will the author be liable for any damages whatsoever arising from the use of this software.

VB Comm 2.0 / Win 3.1

Communications Control for Visual Basic by Mark Gamber August 1992 Version 2.0

VBCOMM provides a communications module for Visual Basic. TO begin using the control, copy VBCOMM.VBX to your Visual Basic directory and select the "File_AddFile" menu item to display a listbox of files. Select VBCOMM.VBX to add the file to the toolbox.

VBCOMM only uses Windows communications calls and is thus subject to policing of ports by Windows. In any given Windows session, anything can and usually go wrong with communications, so no guarantees are implied. That out of the way...

First step in using VBCOMM is to set up the port by setting property values. Once a port is open, the properties should not be altered until the port is closed again. You may initialize the port by setting the following properties:

Comport: Select from the following values:

- 0 - COM1:
- 1 - COM2:
- 2 - COM3:
- 3 - COM4:

Baud: Select from the following values:

- 0 - 300
- 1 - 1200
- 2 - 2400
- 3 - 9600
- 4 - 19200

Parity: Select from the following values:

- 0 - Parity off
- 1 - Parity even
- 2 - Parity odd

Data: Select from the following values:

- 0 - 6 bits
- 1 - 7 bits
- 2 - 8 bits

Stop: Select from the following values:

- 0 - 1 bits
- 1 - 2 bits

To open the port, set the "Enable" property to a non-zero number. The number should remain non-zero until the port is closed by setting "Enable" to zero. When the port is open, incoming characters will cause the "InQueue" routine to be executed. Within this routine you can use DataChr and DataStr to

retrieve characters from the receiving queue.

Enable: 0 to close port, non-zero to open port.

DataChr: Sends and receives byte data values (0-255)

DataStr: Sends and receives data strings.

The zipfile contains Visual Basic example code using the VBCOMM module. For more information on actual use, please look at the source code.

This software is public domain and may be freely distributed. As public domain, the software is not officially supported. In using the software, the user assumes responsibility for proper use of the software and the author is not liable. Period. If the user is unable to comply, the user should destroy the software immediately. The author may be reached on the following services:

America Online E-Mail: PCA MarkG

Compuserve Mail: 76450,2754

Internet Mail: pcamarkg@aol.com

WHelpLib.DLL

WHELP LIB.DLL - Add-On library for Windows Help by Mark Gamber

Version 1.1 7/22/92

Overview:

The Windows 3.1 help subsystem is stocked with a number of macros, none more important than RegisterRoutine() which allows the developer to add DLL functions to the WinHelp system. Once registered, the DLL functions may be called in the same manner as an internal macro. Unfortunately, I'm either missing something or not all parameter interpretation is correct when passing certain types of parameters. Thus, this DLL was born to support wave files

and bitmap display in dialog boxes. This software is completely free and may be freely distributed with your help files. This also entails that the software is not supported per se and

it is YOUR responsibility to use it correctly. If you are the type who wants everything now and for free or likes to sue anybody possible over every misfortune that life can bring, this is not for you. End of story.

With that out of the way, onto the functions.....

Functions:

PlayWaveFile(WaveFilename)

This function plays a wave file specified as WaveFilename. The play type is asynchronous. This function requires Windows version 3.1 or greater. If the function is called from a Windows 3.0 system, it is ignored.

DisplayBmpModal(HelpTitle, BitmapFilename, DialogTitle)

This function displays a bitmap in a modal dialog box. The help system is halted until the dialog box is closed through the system menu. A valid BMP file is passed as BitmapFilename and may be a 2, 16 or 256 color bitmap. You must also provide the title of the Help window and the text to appear in the title bar of the dialog box. For example, if the title of the Help window at

the time the function is called is "Help Test", you would then call the function in the following manner:

DisplayBmpModal("Help Test", "BMPFILE.BMP", "Dialog Title")

DisplayBmpModeless(HelpTitle, BitmapFilename, DialogTitle)

Same as DisplayBmpModal() except the bitmap is displayed in a modeless dialog box.

KillModelessDlg(DlgTitle)

Destroys a dialog box created using DisplayBmpModeless(). DlgTitle is the text which appears in the title

bar of the dialog box as passed to the dialog using the DialogTitle parameter of DisplayModelessBmp().

ChangeDlgBitmap(DlgTitle, NewFilename)

This changes the bitmap to display within a dialog box without destroying the dialog box. If the dialog box cannot be found or the bitmap file does not exist, the function is ignored.

ChangeDlgTitle(DlgTitle, NewDlgTitle)

This changes the title of the dialog box to a new text string. If the dialog box specified cannot be found, the function is ignored.

Usage:

To use either function, they must be registered with the help system using RegisterRoutine() in the [CONFIG] section of the project file (HPJ). For example, to register both functions, play a wave file a display a bitmap when the help file is first opened, you could do this in your HPJ file:

[CONFIG]

```
RegisterRoutine( "WHELP LIB", "PlayWaveFile", "S" )
RegisterRoutine( "WHELP LIB", "DisplayBmpModal", "SSS" )
RegisterRoutine( "WHELP LIB", "DisplayBmpModeless", "SSS" )
RegisterRoutine( "WHELP LIB", "KillModelessDlg", "S" )
RegisterRoutine( "WHELP LIB", "ChangeDlgBitmap", "SS" )
RegisterRoutine( "WHELP LIB", "ChangeDlgTitle", "SS" )
```

```
PlayWaveFile( "TADA.WAV" )
DisplayBmpModal( "Help Title", "WINLOGO.BMP", "Dialog Title" )
DisplayBmpModeless( "Help Title", "WINLOGO.BMP", "Dialog Title" )
```

The RegisterRoutine() calls MUST be made within the [CONFIG] section of the project file. The calls, however, may be anywhere in the help files that normal macros may be called from. For example, after segmenting a bitmap using C 7's SHED utility, you may supply a routine provided which is then called when the bitmap segment is clicked on. You only need to register the routines you intend to use. If you only want to play wave files, for example, you don't need to register any of the dialog functions.

About the Software:

Being in the public domain, this software is not supported per se. I am, however, always willing to listen to ideas and correct bugs if time permits. Since time usually doesn't permit, I restrict the methods of getting in touch with me to the following services:

America Online: E-Mail to PCA MarkG (PC Development)
Internet: E-Mail to pcamarkg@aol.com

No letters, telephone calls or BBS messages or mail are accepted.

Text Box routines

CFGTEXT.TXT

Routines to configure a text box for max length and upper/lower case translation and password hiding

These routines also appear to work fine with QEVb (Pioneer) controls

Jim McClure
QED, Inc.
9/16/92
CSID = 76666,1303
or jamcclure@qed.com from Internet

'Win API Calls

'Put these in your Global module

'-----

Declare Function GetWindowLong& Lib "User" (ByVal hWd%, ByVal nIndex%)

Declare Function SetWindowLong& Lib "User" (ByVal hWd%, ByVal nIndex%, ByVal dwNewLong&)

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

Global Const GWL_STYLE = -16

Global Const ES_UPPERCASE = &H8&

Global Const ES_LOWERCASE = &H10&

Global Const ES_PASSWORD = &H20&

Global Const WM_USER = &H400

Global Const EM_LIMITTEXT = WM_USER + 21

Global Const EM_SETPASSWORDCHAR = WM_USER + 28

'-----

'Get the VBHWND.ZIP file from Lib 1 and use the CTLHWND DLL in it

'-----

Declare Function ControlhWnd% Lib "CTLHWND" (Ctl as Control)

'-----

'Routine to configure a text box

'Put this routine in a general module

'-----

Sub ConfigTextBox (t As Control, Style As Long, MaxLen As Integer)

Dim CtlStyles as Long, NewStyles as Long, Ok as Long

Dim tWnd as Integer

```

'Get handle to text box control
tWnd = ControlhWnd(t)

'Get current style settings
CtlStyles = GetWindowLong(tWnd, GWL_STYLE)

'Add in new styles
'If you want to be able to "toggle" styles, use XOR here and send '1' bits for styles
NewStyles = CtlStyles Or Style

'Set new style
CtlStyles = SetWindowLong(tWnd, GWL_STYLE, NewStyles)

'Set type of password char
'Just use default '*', or could pass in a char for use
If (NewStyles And ES_PASSWORD) > 0 Then
    Ok = SendMessage(tWnd, EM_SETPASSWORDCHAR, Asc("*"), 0&)
End If
End Sub

'-----
'Sample calls to ConfigTextBox
'-----
Sub Form_Load()
    ConfigTextBox Text1, ES_UPPERCASE, 10
    ConfigTextBox Text2, ES_UPPERCASE Or ES_PASSWORD, 10
    'etc...
End Sub

```



ToolBar

The new VBToolBox toolbar adds 8 buttons to Visual Basics existing one. The purpose of this extra toolbar is to add those functions that MS neglected in the original. The left two buttons are user configurable for either Sendkey or Shell functions. The most important feature of this toolbar is that it will always float on top of code windows. Now you can edit in a maximized code window and have access to your editing functions. VBToolBox will monitor your programming environment to integrate closely with VB 2.0. The toolbar will recede in run mode, when a help file or program unrelated to VB is active and minimize or exit in conjunction with Visual Basic.

Note:

Cut, Copy, Paste and instant watch functions etc.. in the floating toolbar function only in design and or break modes, just like their menu counterparts. I cannot foresee what functions you will program into the user configurable buttons, therefore I cannot disable a button if that function is unavailable in one of VB's three modes. If you press a button and nothing happens,

For information on toolbar configuration and positioning also see:

Configuration



Btn #1:

User configurable, This button can be configured to sendkeys to activate menu functions or for the shell function to launch any program.



Btn #2:

User configurable, This button can be configured to sendkeys to activate menu functions or for the shell function to launch any program.



Btn #3:

"Cut" Deletes any selected text and copies to the clipboard.



Btn #4

Copies any selected text to the clipboard.



Btn #5:

Save and Run; This button will save your project and then place VB in run mode. Use this in place of the run command and you will never again loose code if the system locks up or GPF's. For added convenience I have added a global hot key to this feature [Ctrl + F5]. You now have three hot keys for running your programs, F5 for Run; Shift F5 to continue from break mode and Ctrl + F5 for Save Project and Run.



Btn #6:

Auto Watch: The instant watch button MS provided is anything but instant. It calls up two confirmation dialog boxes. If you use the defaults this button will set your watch variables in one click. For keyboard users this function also has a global hotkey. [Ctrl+F9].

MS instant watch = Shift+F9

VBToolBox instant watch = Ctrl+F9



Btn #7

Print selected text. In design or break mode use the mouse to select any portion of your code and this function will copy it to the clipboard and print. [Global hotkey = F6]



Btn #8:

Minimizes the toolbar and turns off floating property.

Print Preview Example

'DRWSCR08.TXT

'DrawScript Routines

'Version 0.8

'9/19/92

,

'Jim McClure (76666,1303)

,

'These routines are designed to simplify the programming of a Print Preview capability for '

'text-oriented reports. The routines record the sequence of Prints, Tabs, etc., for later

'playback to one or more objects (e.g., Printer object, picture control, etc.). The routines also

'provide pagination with header/footer control.

,

'That's the good news. Now for the bad news. I'm still struggling with finding a good way to

'scale the text output for screen display so that it matches printer output. Currently, a point size

'of 8 is used for screen display because of some problems with printing sizes < 8.

'I will upload a revised display strategy later. For now, just use a picture control that scrolls vert.

'and horiz. A point size of 8 is more easily readable anyway!

,

'WARNING!

'This code is still very much under development! I will be uploading revised versions

'periodically if there's enough interest. I will also try to upload an entire mini-project next,

'showing how to use the routines. In the meantime, EXPECT BUGS! Feel free to make

'enhancements, etc. (e.g., adding better line/box support would' be nice). I'll be happy to

'share whatever improvements I make...

,

'But first I have to get some sleep! <g>

,

,

'-----

'Here is some example usage of the routines

,

,

'First, provide your own routine called "dsBoundaryPrint" to

' print headers and footers (boundaries) as needed. Your

' dsBoundaryPrint routine will be called as follows:

,

' Sub dsBoundaryPrint(Region as integer, PageNum as integer)

,

'In your routine, you can use dsPrint, dsTab, etc., calls to print

' a nice header (if Region = 1) or footer (if Region = 2) for

' your report. Just be sure to print the same # of lines that

' you specify in the dsNew() function below.

,

'Create a new draw script for output page size of 60 lines,

```

' with 5 lines reserved for the header and 5 lines reserved for the
' footer, using the base font "Helv" point size 12:
',

' hDS% = dsNew(60, 5, 5, "Helv", 12)
',

'Print a few things to it:
',

' dsPrintNL "Hello World!"
' dsTab 30
' dsPrintNL "This is indented!"
' dsFontUnderLine TRUE
' dsPrintNL "This is underlined!"
' dsFontUnderLine FALSE
' dsPrintAttr "This is also underlined!", "U"    'U = underline
' dsNL 'This finishes prior line
' dsLine 'This draws a simple separator line on the output
' (NOTE: The separator doesn't take up a "line" of output-- it leaves
'   the print cursor where it is.)
',

'Ok, we're done formatting
' dsClose(hDS%)
',

'Find out how many pages were generated
' nPages = dsMaxPages()
',

'(Remember, each page will have the appropriate header/footer
' provided by your dsBoundaryPrint routine.)
',

'Play them all back to the printer, starting at page #1
' dsPlay hDS%, DummyControl, TRUE, 1, nPages 'TRUE=Send to printer
',

'Play one page of same report back to a picture box-- start at page #3 this time
' dsPlay hDS%, RealPictureControl, FALSE, 3, 1
'(Now, set up a scroll bar or set of buttons to keep calling
' dsPlay with a larger PageStart, or allow user to jump directly
' to page # by entering it)
',

'Ok, don't need this draw script anymore
' dsFree(hDS%)
'(If you don't do this, a temp file will be left behind!)
',

'GOOD LUCK!
' Jim
',
'-----

```

'This goes in your Global.Bas module

'DrawScript data structure

Type DrawScriptType

Alloc As Integer

FileNum As Integer

FileName As String

MaxLines As Integer

HeaderLines As Integer

FooterLines As Integer

CurLine As Integer

CurPage As Integer

MaxPages As Integer

End Type

'-----

'This can go in a module called DrawScript.Bas

'Allocate array of DrawScript structures

Const nDrawScripts = 5

Dim DrawScript(nDrawScripts) As DrawScriptType

'The following hold the 'current' DS

Dim dsCurrent As Integer

Dim dsFileNum As Integer

Dim dsMaxLines As Integer, dsHeaderLines As Integer, dsFooterLines As Integer

Dim dsCurLine As Integer

Dim dsInBoundary As Integer

Dim dsCurPage As Integer, dsMaxPageNum As Integer

'-----

'Here come the routines

Sub dsPrint (PrintString As String)

'Print a string to the current DS

,

'Process header/footer

If Not dsInBoundary Then

dsCheckBoundary

End If

'Print string

Print #dsFileNum, "PR " + PrintString

End Sub


```
Sub dsPlay (hDS As Integer, c As Control, ToPrinter As Integer, PageStart As Integer, NumPages As Integer)
```

```
    'Replay draw script on output device
```

```
    'Either the Printer object (if ToPrinter is true)
```

```
    ' or to the supplied control "c" (e.g., form, picture)
```

```
    'Replay starts at PageStart (1st page = 1) and
```

```
    'proceeds for NumPages pages
```

```
    '
```

```
    Dim InpString As String, Cmd As String, Arg As String
```

```
    Dim FileNum As Integer, StopNow As Integer
```

```
    Dim PageCount As Integer
```

```
    'Get a file number for use
```

```
    FileNum = FreeFile
```

```
    'Open the file for processing
```

```
    Open DrawScript(hDS).FileName For Input As #FileNum
```

```
    'See to starting page
```

```
    PageCount = 1
```

```
    Do While (PageCount < PageStart) And (Not EOF(FileNum))
```

```
        'Read each line from the file
```

```
        Line Input #FileNum, InpString
```

```
        'Increment page count
```

```
        If Left$(InpString, 2) = "NP" Then
```

```
            PageCount = PageCount + 1
```

```
        End If
```

```
    Loop
```

```
    'Process file 'till end
```

```
    StopNow = FALSE
```

```
    Do While (Not EOF(FileNum)) And (Not StopNow)
```

```
        'Read each line from the file
```

```
        Line Input #FileNum, InpString
```

```
        'Separate command from data
```

```
        Cmd = Left$(InpString, 2)
```

```
        If Len(InpString) > 3 Then
```

```
            Arg = Right$(InpString, Len(InpString) - 3)
```

```
        Else
```

```
            Arg = ""
```

```
        End If
```

'Depending on which command is present...

Select Case Cmd

Case "PR"

'Print a string

If ToPrinter Then

Printer.Print Arg;

Else

c.Print Arg;

End If

Case "NL"

'Start a new line

If ToPrinter Then

Printer.Print

Else

c.Print

End If

Case "TB"

'Tab to specified location

If ToPrinter Then

Printer.Print Tab(Val(Arg));

Else

c.Print Tab(Val(Arg));

End If

Case "LN"

'Draw separator line

If ToPrinter Then

Printer.Line -Step(Printer.ScaleWidth, 0)

Printer.CurrentX = 0

Else

c.Line -Step(c.Width, 0)

c.CurrentX = 0

End If

Case "FB"

'Set FontBold property

If ToPrinter Then

Printer.FontBold = Val(Arg)

Else

c.FontBold = Val(Arg)

End If

Case "FU"

'Set FontUnderline property

If ToPrinter Then

Printer.FontUnderline = Val(Arg)

Else

c.FontUnderline = Val(Arg)

```

    End If
Case "FI"
    'Set FontItalic property
    If ToPrinter Then
        Printer.FontItalic = Val(Arg)
    Else
        c.FontItalic = Val(Arg)
    End If
Case "FS"
    'Set FontStrikethru property
    If ToPrinter Then
        Printer.FontStrikethru = Val(Arg)
    Else
        c.FontStrikethru = Val(Arg)
    End If
Case "FZ"
    'Set FontSize property
    If ToPrinter Then
        Printer.FontSize = Val(Arg)
    Else
        'Scale font size for screen
        c.FontSize = 8
    End If
Case "FN"
    'Set FontName property
    If ToPrinter Then
        Printer.FontName = Arg
    Else
        c.FontName = Arg
    End If
Case "NP"
    'Start new page
    If ToPrinter Then
        Printer.NewPage
    End If

    'Keep track of # of pages
    PageCount = PageCount + 1

    'See if we should quit
    If (Not ToPrinter) Or (PageCount = PageStart + NumPages) Then
        StopNow = TRUE
    End If
End Select
Loop

```

```

'Done with file
Close #FileNum

'If done with printer, close it
If ToPrinter Then
    Printer.EndDoc
End If
End Sub

Function dsNew (MaxLines As Integer, nHeader As Integer, nFooter As Integer, FontName As String,
FontSize As Integer) As Integer
    'Returns a handle to a DrawScript structure
    ' or -1 if unable to alloc another structure
    'NOTE: This command does an implicit dsSet() of the
    ' new hDS
    '
    Dim hDS As Integer, i As Integer

    'Look for a free descriptor
    hDS = -1
    For i = 0 To nDrawScripts - 1
        If Not DrawScript(i).Alloc Then hDS = i
    Next i

    'If we could allocate an element
    If hDS >= 0 Then
        'Remember that this element is allocated
        DrawScript(hDS).Alloc = TRUE

        'Get a new file descriptor
        DrawScript(hDS).FileNum = FreeFile

        'Setup filename
        '(Might want to put these files into TEMP dir)
        DrawScript(hDS).FileName = "DSTEMP" + LTrim$(Str$(hDS)) + ".TXT"
        Open DrawScript(hDS).FileName For Output As #DrawScript(hDS).FileNum

        'Set initial font name and size
        Print #DrawScript(hDS).FileNum, "FN " + FontName
        Print #DrawScript(hDS).FileNum, "FZ" + Str$(FontSize)

        'Set rest of data
        DrawScript(hDS).MaxLines = MaxLines
        DrawScript(hDS).HeaderLines = nHeader
    End If
End Function

```

```

        DrawScript(hDS).FooterLines = nFooter
        DrawScript(hDS).CurLine = 0
        DrawScript(hDS).CurPage = 0
        DrawScript(hDS).MaxPages = 0

        'Set current hDS
        dsSet hDS
    End If

    'Return the desired descriptor
    dsNew = hDS
End Function

Sub dsClose (hDS As Integer)
    'Finished outputting drawing commands to this DS

    'First, finish the current page
    If dsCurLine > 0 Then
        dsNewPage
    End If

    'Just close the file
    Close #DrawScript(hDS).FileNum

    'Remember the # of pages
    dsMaxPageNum = dsCurPage
End Sub

Sub dsPrintNL (PrintString As String)
    'Process header/footer
    If Not dsInBoundary Then
        dsCheckBoundary
    End If

    'Print string and start new line
    Print #dsFileNum, "PR " + PrintString
    Print #dsFileNum, "NL"
    dsCurLine = dsCurLine + 1
End Sub

Sub dsTab (Col As Integer)
    'Process header/footer
    If Not dsInBoundary Then
        dsCheckBoundary
    End If

```

```
'Tab to desired position in output
Print #dsFileNum, "TB " + LTrim$(Str$(Col))
End Sub
```

```
Sub dsLine ()
'This routine draws a single horizontal separator line
'
'Process header/footer
If Not dsInBoundary Then
    dsCheckBoundary
End If

'Just save the command for later
Print #dsFileNum, "LN"
End Sub
```

```
Sub dsFree (hDS As Integer)
'Done with a draw list-- free it
'First, remove the temp file
Kill DrawScript(hDS).FileName

'Now, mark the descriptor as free
DrawScript(hDS).Alloc = FALSE
End Sub
```

```
Sub dsFontUnderline (Value As Integer)
'Process header/footer
If Not dsInBoundary Then
    dsCheckBoundary
End If

Print #dsFileNum, "FU" + Str$(Value)
End Sub
```

```
Sub dsFontBold (Value As Integer)
'Process header/footer
If Not dsInBoundary Then
    dsCheckBoundary
End If

Print #dsFileNum, "FB" + Str$(Value)
End Sub
```

```
Sub dsFontItalic (Value As Integer)
```

```

'Process header/footer
If Not dsInBoundary Then
    dsCheckBoundary
End If

Print #dsFileNum, "FI" + Str$(Value)
End Sub

Sub dsFontStrikethru (Value As Integer)
'Process header/footer
If Not dsInBoundary Then
    dsCheckBoundary
End If

Print #dsFileNum, "FS" + Str$(Value)
End Sub

Sub dsFontSize (Size As Integer)
'Process header/footer
If Not dsInBoundary Then
    dsCheckBoundary
End If

Print #dsFileNum, "FZ" + Str$(Size)
End Sub

Sub dsNL ()
'Force a new line
,

'Process header/footer
If Not dsInBoundary Then
    dsCheckBoundary
End If

'Start new line
Print #dsFileNum, "NL"
dsCurLine = dsCurLine + 1
End Sub

Sub dsNewPage ()
'Force a new page on the current DS
,

Dim i As Integer

'Check header/footer

```

If Not dsInBoundary Then

 'Generate header, if needed

 'See if we're in the header region

 If dsCurLine = 0 Then

 'If not 1st page

 If dsCurPage > 0 Then

 'Start new page

 Print #dsFileNum, "NP"

 End If

 'Starting new page

 dsCurPage = dsCurPage + 1

 'Activate the header

 dsInBoundary = TRUE

 dsBoundaryPrint 1, dsCurPage

 dsInBoundary = FALSE

End If

 'Skip as many lines as are needed

 For i = dsCurLine To dsMaxLines - dsFooterLines - 1

 dsNL

 Next i

 'Now generate footer

 'See if we're in the footer region

 If dsCurLine = dsMaxLines - dsFooterLines Then

 'Activate the footer

 dsInBoundary = TRUE

 dsBoundaryPrint 2, dsCurPage

 dsInBoundary = FALSE

 End If

End If

 'Reset line count

 dsCurLine = 0

End Sub

Sub dsFontName (FontName As String)

 'Process header/footer

 If Not dsInBoundary Then

 dsCheckBoundary

 End If


```
    Print #dsFileNum, "FN " + FontName
End Sub
```

```
Sub dsTabPrint (Col As Integer, S As String)
    'Process header/footer
    If Not dsInBoundary Then
        dsCheckBoundary
    End If
```

```
    'Tab to spec location and print string
    Print #dsFileNum, "TB" + Str$(Col)
    Print #dsFileNum, "PR " + S
End Sub
```

```
Sub dsTabPrintNL (Col As Integer, S As String)
    'Process header/footer
    If Not dsInBoundary Then
        dsCheckBoundary
    End If
```

```
    'Tab to spec location and print string, following by newline
    Print #dsFileNum, "TB" + Str$(Col)
    Print #dsFileNum, "PR " + S
    Print #dsFileNum, "NL"
    dsCurLine = dsCurLine + 1
End Sub
```

```
Sub dsPrintAttr (PrintString As String, Attrs As String)
    'Print string with specified attributes
    ' e.g., "U" = underline
    '       "B" = bold
    '       "S" = strikethru
    '       "I" = italic
    '
    Dim i As Integer

    'Process header/footer
    If Not dsInBoundary Then
        dsCheckBoundary
    End If

    'Set each attribute
    For i = 1 To Len(Attrs)
        Print #dsFileNum, "F" + Mid$(Attrs, i, 1) + " -1"
```

```

Next i

'Print the desired string
Print #dsFileNum, "PR " + PrintString
'Turn off the attributes

For i = 1 To Len(Attrs)
    Print #dsFileNum, "F" + Mid$(Attrs, i, 1) + " 0"
Next i
End Sub

'This routine may not be fully working yet.
'It's supposed to allow you to have several DS going at once
' and be able to switch between them. Handy for slow DBMSs! (Format
' several reports on the same data at once.)
Sub dsSet (hDS As Integer)
    'Save current DS
    DrawScript(dsCurrent).FileNum = dsFileNum
    DrawScript(dsCurrent).MaxLines = dsMaxLines
    DrawScript(dsCurrent).HeaderLines = dsHeaderLines
    DrawScript(dsCurrent).FooterLines = dsFooterLines
    DrawScript(dsCurrent).CurLine = dsCurLine
    DrawScript(dsCurrent).CurPage = dsCurPage
    DrawScript(dsCurrent).MaxPages = dsMaxPageNum

    'Set new hDS for subsequent calls
    dsCurrent = hDS
    dsFileNum = DrawScript(hDS).FileNum
    dsMaxLines = DrawScript(hDS).MaxLines
    dsHeaderLines = DrawScript(hDS).HeaderLines
    dsFooterLines = DrawScript(hDS).FooterLines
    dsCurLine = DrawScript(hDS).CurLine
    dsCurPage = DrawScript(hDS).CurPage
    dsMaxPageNum = DrawScript(hDS).MaxPages
End Sub

Sub dsCheckBoundary ()
    'This routine checks to see whether we've
    'come to a boundary (header or footer)
    'in the report
    '
    'See if we're in the footer region
    If dsCurLine = dsMaxLines - dsFooterLines Then
        'Activate the footer
        dsInBoundary = TRUE
    
```

```

    dsBoundaryPrint 2, dsCurPage
    dsInBoundary = FALSE

    'Reset line count
    dsCurLine = 0
End If

'See if we're in the header region
If dsCurLine = 0 Then
    'If not 1st page
    If dsCurPage > 0 Then
        'Start new page
        Print #dsFileNum, "NP"
    End If

    'Starting new page
    dsCurPage = dsCurPage + 1

    'Activate the header
    dsInBoundary = TRUE
    dsBoundaryPrint 1, dsCurPage
    dsInBoundary = FALSE
End If
End Sub

Function dsMaxPages () As Integer
    'This function returns the max # of pages for the current DS
    dsMaxPages = dsMaxPageNum
End Function

'-----
'Here is a sample boundary print routine
'
' Remember, YOU supply this, so you can put whatever you
' want in it for titles, etc.
'
Sub dsBoundaryPrint(Region as Integer, PageNum as Integer)
    Select Case Region
        case 1 ' Header
            dsNL
            dsPrintAttr "Quarterly Report", "B" 'Show title in BOLD
            dsNL ' Finish prior line
            dsNL
        case 2 ' Footer

```

```
    dsNL
    dsTabPrintNL 40, PageNum ' Show page # on footer
    dsNL
End Select
End Sub
```