VBXRef

Version 1.16

Introduction

Getting Started

Starting VBXRef

Menu Selections

File Operations
Reports
Options
Help

Using VBXRef

Error Codes

Whenever an error occurs during the operation of VBXRef, a warning appears giving you a short descriptive message and a unique error number. For more information about this particular error, press the "Help" Command Button.

Commonly Asked Questions

Question: What does an "unexpected error" message mean?

Answer: This message can indicate that the DLL used with your program is not compatible with the version of Visual Basic being used. If you have a version of VB that is later than 1.00, you may need to contact MicroHelp for an upgrade.

This message can also indicate a failure to locate the DLL. Make sure that the DLL resides in the user's PATH.

Question: I have some questions about Visual Basic programming, but they have nothing to do with a MicroHelp product. What should I do?

Answer: Microsoft technical support is best equipped to answer questions about VB programming. In addition, Microsoft maintains a "knowledge base" on CompuServe (GO MSKB) where you'll find a wealth of information about Basic programming.

Question: I'd like to learn more about the Windows API. Where can I get information on the routines it contains?

Answer: The "bible" on the subject of the Windows API is the Microsoft Windows Programmer's Reference book, which is published by Microsoft Press. It contains detailed information on every API function call, record structure and constant used in Windows 3.

Be prepared to learn a little bit about the C language if you get a copy of this book. The book, like Windows itself, is built around C. One of the most important aspects of calling API functions is passing the proper parameters. When you use this book, you'll have to translate the C data types to their Visual Basic equivalents.

Alternatively, Microsoft provides a file, called WINAPI.ZIP, on CompuServe (currently in the MSBASIC forum, LIB 5). This compressed file contains an on-line copy of the programmer's reference manual and a text file with the Declare instructions for many of the API routines converted to Visual Basic already.

Question: How can I write Custom Controls like the ones in VBTools? Do I have to use Assembly Language or can I use Visual Basic?

Answer: While you can write Custom Controls in Assembly, the best language to use is C. To create your own Custom Controls, you'll need to get a copy of the Microsoft Control Development Kit (CDK). Additionally, you'll need Microsoft C version 6.0 and the Microsoft Software Development Kit (SDK) or the Microsoft QuickC for Windows.

You can't create Custom Controls using Visual Basic.

Glossary

API -- Applications Program Interface. The method by which the programmer can invoke Windows service routines and functions that exist in the Windows Dynamic Link Libraries (DLL).

Bitmap -- In general, the term bitmap refers to an image formed by a pattern of bits, rather than a pattern of lines. In Microsoft Windows, there are two kinds of bitmaps:

> A "device-dependent" bitmap is a pattern of bits in memory which can be displayed on an output device. Because there is a close correlation between the bits in memory and the pixels on the display device, a memory bitmap is said to be device dependent. For such bitmaps, the way the bits are arranged in memory depends on the intended output device.
>
> A "device-independent" bitmap (DIB) describes the actual appearance of an image, rather than the way that image is internally represented by a particular display device. Because this external definition can be applied to any display device, it is referred to as device independent.

Caret -- Refers to the vertical bar that indicates the position where text is overwritten or inserted. In other words, the Caret is the equivalent of the Cursor as used in other versions of BASIC.

Cell -- The area at the intersection of a row and column in a list box control.

Child form/window -- Refers to a form that is inside another form. Please see the information on the Multiple Document Interface for more information.

Cursor -- Refers to the mouse cursor.

Custom colors -- Please see the term "MicroHelp standard color properties".

Custom controls -- These are controls you add to your programs when you want to go beyond the capabilities of the controls that are a part of Visual Basic.

Custom DLL -- In addition to callable routines, custom DLL's contain resources.

Custom event -- Refers to an event that only works with a VBTools Custom Control.

Custom properties -- In order to provide enhanced functionality, many of the Custom Controls contain "custom" properties that are not present in standard VB controls.

DLL -- Dynamic Link Library. An executable module containing functions that Windows applications can call in order to perform useful tasks. (If you have used Microsoft QuickBASIC 2 or later, you can think of DLL's in terms of "User" or "Quick" or "Extended Runtime" libraries, or even the "runtime" libraries you use with CHAIN.)

Extended selection -- Refers to the way multiple items can be selected in File Manager, i.e., a *range* of items can be selected. The .ExtendedSelect property allows the selection of multiple *contiguous* items. See also Single selection.

Fixed -- An attribute used to describe List Box rows or columns that don't scroll.

Inner areas -- A general term used to refer to rectangular regions inside controls where a caption or a picture can be displayed. The inner area is defined by the Left, Top, Bottom and Right positions, relative to the control's borders.

MDI -- The Multiple Document Interface (MDI) is a Windows standard that specifies the way in which applications can provide access to multiple forms within a form.

Mouse Sizing -- Refers to the ability to size the "inner" areas of many controls using the mouse in design mode. See Section 11(E) for details.

Parent -- Refers to an object (form or control) in which a control resides. Please see the information on the Multiple Document Interface for more information.

PCX -- A file containing graphics information in a format originally created by ZSoft Corporation. The MhPicture control can read PCX files.

Property array -- Refers to a subscripted property you use to address individual elements of a control. For example, most of the custom List Box controls contain a property array, named .Tagged, that tells you if a line has been selected. Each element of the .Tagged property array corresponds to a line in the List Box.

Range -- A term used to describe a group of cells. Typically, a range is denoted by its upper-left and lower-right locations.

Resource -- Bitmaps, custom controls, custom cursors, etc.

SDK -- Microsoft Windows Software Development Kit

Single selection -- Refers to whether the control can be used to select multiple, non-contiguous items. The .SingleSelect property sets or returns this value in many Custom Controls. See also ExtendedSelect.

State buttons -- These buttons toggle from one *state* to another when you click them. They stay in the new state till you click them again. The two states are raised and depressed. Normal buttons are depressed when you click them with the mouse and raised when you release the mouse button.

Thumb -- The indicator on a scrollbar that represents an approximate relative position. This is the portion of the scrollbar that moves when you click the arrows and that you drag with the mouse.

Other MicroHelp Products for Visual Basic

VBTools version 2- Contains more than 30 custom controls, video "special effects", examples of how to use some of the more popular Windows API functions, access to the Windows 3.1 "Common Dialog Routines" and much MORE!

MicroHelp Muscle - The only Visual Basic add-on to win Computer Language Magazine's 1991 Productivity Award. Muscle contains over 430 assembly language routines that provide you with the speed and functionality that is not available anywhere else. DOS and Windows low level services, string and array manipulation, date and time functions, bit manipulation and routines that will allow you to create arrays that can use all available Windows memory (including temporary and permanent swap files).

MicroHelp Communications Library - Since Visual Basic does not have any native communications support, this library provides you with everything you need, including source code for a working Windows terminal program. Seven different file transfer protocols that work even when your application is minimized, ANSI terminal emulation, and the ability to have upto eight simultaneous communications sessions at ONE TIME!

MicroHelp Network Library - 100% assembly language routines that support Novell, Lantastic and NetBIOS API functions. Now you can access and manipulate all objects in the bindery, add, change and delete items in the print queues as well as redirect all shared resources.

For more information and free DEMOS call:

1-800-922-3383

***In Georgia or Outside the U.S. (404) 594-1185**

Documentation Conventions

All MicroHelp manuals/help files use various type styles to illustrate key points in the text. Section headings and other MicroHelp products are shown in **bold**. Items requiring special emphasis are printed in ***underlined italics***. References to other sections of this manual are printed in *plain italics*. A program fragment is shown below. Note the use of the arrow symbol ⇒to indicate that the code was too long to fit on the previous line.

```
Main Program:'Comments are shown out to the  PRINT "HELLO" 'side when possible
and are always
  A% = 10       'preceded with an apostrophe "'".
  PRINT "This line is too long and overlaps onto
  ⇒ the next line"
```

When it is not possible to keep the comments on the right of the sample source code listing, they are placed starting with the current indentation level and shown in the most readable manner.

Product Description

VBXRef is a Visual Basic application that provides you with detailed information about the Forms and modules in your Visual Basic programs. The program is designed to simplify project management and help in debugging large and complicated programs.

With the reports generated by VBXRef, you can quickly determine what controls are used in your project, their current property values and which values have been changed from the defaults. You can also generate a detailed cross reference of all sub procedures, functions, scalar variables, arrays, form and control properties - even string and numeric literals - as well as all of the Visual Basic keywords used in the project. VBXRef will detail the number of occurrences of each of these items, as well as every module and procedure in which they are referenced.

Prior to VBXRef there was no way you could print out or view, from outside the Visual Basic environment, all of the controls with their properties and values along with the source code. Nor was there any program available that could print the form and control information in conjunction with your source code.

VBXRef was created using custom controls and assembly language routines from our two top-selling Visual Basic add-ons, **VBTools** 2.1 and MicroHelp **Muscle** for Visual Basic. Without these two tools, an application like VBXRef would be impossible to create with Visual Basic alone. MicroHelp offers more add-on products for Visual Basic than any other vendor, and our **Muscle** product won the 1991 Computer Language magazine's Productivity Award, the *only* add-on product for Visual Basic to be selected.

Technical Support

Technical support for VBXRef is available to the registered user, Monday - Friday 9AM to 5PM Eastern time. Our technical support telephone number is (404) 516-0898. Before you call for technical support, please read, *Commonly Asked Questions*. While you are welcome to FAX technical support questions, we respond to all written support questions (including FAX's) via US Mail.

If the information under Commonly Asked Questions does not help you resolve the problem, we will be happy to do our best to help you. Please have your serial number handy (it is on the diskette label and we suggest you write it on the title page of the manual as well), the VBXRef version number (it's listed in the "About Box"), and, if possible, be at your computer when you call. If you are getting an error message from VBXRef or Windows, please note the error number and message and have it ready for our technical support personnel.

If you receive a "UAE" or "General Protection Fault" message, it is not necessary to record address or other information. However, it is important that the problem can be duplicated . Please note the steps taken prior to receiving the message.

One important note about UAE's and GP Faults: Please remember to exit Windows before attempting to duplicate the problem that caused the UAE or GP Fault. Windows can become very unstable after one of these errors and subsequent results are usually unreliable.

Since the file formats for the binary form and module files in Visual Basic have not been published by Microsoft, the reading of these files is not an "exact science". This is also the reason the format of all third parties' custom controls cannot be read. However, regardless of the control involved, VBXRef should never crash or cause a UAE or GP Fault. Your assistance in helping duplicate any of these problems is appreciated, so that VBXRef can be fixed and improved. If you are willing to send us a form that causes one of these problems, please do the following:

1) If the form uses any other third party controls other than MicroHelp products or those in the Microsoft VB Professional Toolkit, please remove these controls if possible. Of course, if one of these controls is related to the problem, leave it on the form. Also include the control data file for this control, the name of the vendor who created the control and their phone number, if possible.

2) If the problem is related to the cross reference module, then you need only include the ASCII files necessary to duplicate the problem.

3) In a separate ASCII text file, please include your name, telephone number, serial number, VBXRef version number and a brief description of the problem and how to duplicate it. Archive the files using either PKZIP or LHARC (both of these popular programs are available on our BBS), send the archived file to us on a diskette or upload it to our Bulletin Board System. The telephone numbers for our BBS are: (404) 516-1497 [HST Dual Standard] and (404) 516-1397 [2400]. Our BBS uses PC Board software and supports all of the popular file transfer protocols. If you need help accessing our BBS, please call our technical support department.

NOTE: The inability of VBXRef to read all of the properties for a third party control is not considered a "bug" in VBXRef and does not require you to send us a form. We are aware of most of the controls that VBXRef cannot currently read and are working on a method to resolve this problem. It is not necessary for you to contact our technical support department whenever you receive an error 100. Instead please follow the steps outlined for the error under *Warning Errors*. For additional information about VBXRef and custom controls, please see *Custom Controls* under Using VBXRef.

Usage and Distribution

VBXRef is a stand alone application that is licensed for use only by the registered user of this product. All of the DLL's and supporting files distributed with this product are meant for use with this product alone and cannot be distributed by the licensee to any other parties.

Compatibility

This product was designed to be compatible with Visual Basic version 1.0. If you have a later version, and you have any problems whatsoever, please contact MicroHelp Technical Support to determine if a more recent version of this product is available.

System Requirements

The minimum system requirements for using VBXRef are:

A computer running Microsoft Windows 3.0 or later with a minimum of 2 MB of RAM.
Microsoft Visual Basic 1.00.
1 Diskette drive (1.2MB 5.25" or 1.44MB 3.5").
A color VGA monitor.

Your Registration Card

In the diskette envelope you will find the product distribution diskette and a registration card. If you would like to be eligible for our full-time technical support and product upgrades at reduced costs, you must fill out the registration card and mail it to us. Without the card on file, our staff has no way of verifying that you are a licensed user of the product.

NOTE: *It is VERY important that you complete the registration card and return it to us as soon as possible.*

Making Backup Copies

Immediately after opening the product container, make a "backup" copy of the distribution diskette for safekeeping. This is most easily accomplished using the DOS "DISKCOPY" (to copy the information from the distribution disk to another diskette). Please refer to your DOS manual if you require instructions for using this command.

Installation

In order to install VBXRef, place the distribution diskette into the floppy disk drive and close the door. Then do the following:

Make the floppy disk drive the default drive. For example, if the floppy drive is A:, you would type:

```
A: <Enter>
```

Select a drive and directory where you want to install the product. We recommend "C:\WINDOWS\VBXREF".

On the distribution diskettes is a file called INSTALL.BAT. This batch file is designed to automatically configure your system

Invoke the batch file, using the drive and directory name that you selected. For example:

```
INSTALL C:\WINDOWS\VBXREF <Enter>
```

The installation batch file will do the following for you:

Create a subdirectory on the drive you specified. This directory will contain the executable program and runtime DLL's.

Extract the contents of the "archive" files on your distribution disk to the new directory.

If your are re-installing this product to a directory containing an older version of the same product, you can use the /d switch to avoid having the installation program ask you for permission to overwrite each file. Using the previous command line as a further example:

```
INSTALL C:\WINDOWS\VBXREF /d<Enter>
```

IMPORTANT: Please refer to *The Path Environment Variable* for important instructions that must be followed before you can use this product with Visual Basic.

IMPORTANT: After you install VBXRef, copy the runtime DLL's to your Windows directory or any other directory in your executable path, but the VBXREF.INI, VBKWORDS.DAT and the control data files must remain in the same directory as VBXREF.EXE.

Files On Your Distribution Diskette

The INSTALL.BAT batch file automatically unpacks all of the files onto your hard disk. Except for INSTALL.BAT, all of the files on the distribution diskette are self-extracting archives that contain the files you will use. After installation, you'll find the following files on your fixed disk:

READ.ME	Post-release info.
VBXREF.EXE	The executable application.
VBXREF.HLP	Windows Help file. Contains most of the same information you'll find in this manual.
VBXREF.INI	Please refer to *The VBXREF.INI file* for more information.
VBKWORDS.DAT	An ASCII file containing a listing of Visual Basic keywords and operators. This file is used when creating cross-reference reports of your projects.
*.DAT	Control data files that allow VBXRef to read the binary .FRM files.
READFILE.SUB	VB source code example to read the cross -reference data files. For more information on this file see *Data File Formats* [Requires MicroHelp **Muscle.**]
MHUTIL.VBX	DLL containing support routines for VBXREF.EXE
MH??200.VBX	Runtime DLL's. from VBTools 2.1. If you have VBTools 2.1, you can use the design mode versions of these files and delete these files from your disk.
COMMDLG.DLL	Common Dialog routines from Microsoft. If you have Windows 3.1, this DLL is already in your \WINDOWS\SYSTEM directory.
DEMPROPC.FRM	An example form that you can use to create control data files.
GETDEC.EXE	A free MicroHelp VB utility that will read your source files (you must save them as text) and create a customized listing of only those declarations and references used in your project.
GETDEC.HLP	A Windows help file for the GetDec utility.
MUSCLE.VBX	Runtime DLL of our MicroHelp Muscle library that is needed by GETDEC.EXE.

READ.ME

In the main directory where you install this product is an ASCII text file named "READ.ME". It contains important information about this product, such as corrections to the manual and other anomalies which may have been discovered after the manual went to press. All updated information will also be placed in the VBXREF.HLP file which will be available with all updates to the executable file.

Acknowledgements

A very special thank you is extended to (in alphabetical order): Robert Arnson, J.D. Evans, Craig Leach, Gregg Morris, Ken Spreitzer, and Jonathan Zuck who went far beyond the call of duty in providing assistance and suggestions during the beta cycle of this product.

Some short definitions passages were extracted from the manual for Microsoft(C) WindowsTM Software Development Kit, Copyright 1985-1990 Microsoft Corporation. Reprinted with permission from Microsoft Corporation.

Trademarks

The following trademarks are used throughout this manual. Whenever you come across them, please remember that they are the trademarks or registered trademarks of the companies shown below.

IBM, IBM PC, IBM PC XT, IBM PC AT, PS/2, and PC-DOS are trademarks of International Business Machines Corporation.

Microsoft, MS and MS-DOS are registered trademarks and Microsoft QuickBASIC, Visual Basic and Windows are trademarks of Microsoft Corporation.

The PATH Environment Variable

In order to use a DLL file (or an .EXE executable file) from within Visual Basic, the file must be accessible either in the current directory, or by referring to the DOS PATH. It is therefore important to do one of three things:

Copy the *.VBX and *.DLL files to a directory in the DOS PATH, or

Copy the *.VBX and *.DLL files to your \WINDOWS\SYSTEM directory, or

Modify your AUTOEXEC.BAT file so that the directory where you installed this product is included in the DOS PATH. For example:

```
SET PATH=C:DOS;C:\WINDOWS;C:\WINDOWS\VBTOOLS;
```

Note each directory is separated from the others by a semicolon character ";".

IMPORTANT: If the DLL files needed by VBXRef are not in the DOS PATH or in the \WINDOWS\SYSTEM subdirectory, Visual Basic will not be able to find them and the program will *not* execute.

Dynamic Link Libraries (DLL)

According to Microsoft, "a DLL is an executable module containing functions that Windows applications can call in order to perform useful tasks". If you have used Microsoft QuickBASIC 2 or later, you can think of DLL's in terms of "User" or "Quick" or "Extended Runtime" libraries, or even the "runtime" libraries you use with CHAIN.

Then there are "custom" DLL's. In addition to callable routines, custom DLL's contain "resources". The term resources refers to bitmaps, custom controls, custom cursors, etc.

VBXRef uses two of the runtime DLL's from **VBTools 2.1 -** MHAD200.VBX and MHEN200.VBX, a DLL containing assembly language routines named MHUTIL.VBX, and the Microsoft Windows 3.1 common dialog DLL named COMMDLG.DLL. The only difference between the VBTools runtime and design time VBX files is that the runtime DLL's can be accessed only from a compiled .EXE program and not from within the Visual Basic program itself.

Loading VBXRef

VBXRef is loaded in the same manner as any Windows application. From Program Manager, select File - Run and then specify the directory and filename of the executable program.
Example:

```
\WIN31\VBXREF\VBXREF
```

We recommend placing VBXRef in a program group where it can be run by simply double-clicking on the icon. For more information on adding an application to a program group, please refer to your Microsoft Windows documentation.

The VBXREF.INI File

When VBXRef loads, it looks for the VBXREF.INI file in the same directory as the executable program VBXREF.EXE. If VBXREF.INI is not found, VBXRef will create a new INI file using the default settings, which are the same values in the VBXREF.INI file on the distribution diskette.

If you examine the VBXREF.INI file, you will notice that it looks similar to your other Windows .INI files. The headings in a Windows INI file that are in brackets [ ] are normally referred to as "application names" since they commonly contain information unique to a specific application. The items to the left of the equal sign "=" underneath the application name entries are called "keynames" and usually identify some function or option in the application. The values or settings for these keynames are to the right of the equal sign. If you examine your WIN.INI file you will probably see several application name entries for most of your Windows programs, some of which may no longer exist on your system!

Since many people do not like the "feature" of some Windows programs to automatically modify their WIN.INI file, we decided to use the more typical DOS application approach of using a program -specific configuration file. This way you don't have to be concerned about which entries in your WIN.INI file may effect certain programs. Everything VBXRef needs for proper configuration and operation is contained in its own INI file.

VBXRef does examine one entry in your WIN.INI file, which is the "COUNTRY = " settting. This entry determines what keystrokes can be used with the *"Save Form File(s) As Text"* option.

Each time that VBXRef is unloaded, it automatically updates the information in your VBXREF.INI file. While this file is a pure ASCII text file that you can modify using Windows Notepad or a similar editor, we recommend that you allow VBXRef to update the file for you. Should you attempt to modify the VBXREF.INI file and VBXRef displays an error message during initialization that it is unable to load this file, you should delete the damaged .INI file and allow VBXRef to create a new one for you.

If you have **VBTools** version 2, the Property Manager control in that product also uses VBXREF.INI to determine which control data files it should load. You can have a separate .INI file for each program, or they can both use same one.

CAUTION: While the VBXREF.INI is structured in a fashion similar to standard Windows INI files, the "Application Name" and "KeyName" values in the VBXREF.INI file *must not be changed.*

The default VBXREF.INI file contains the following lines:

```
[File Viewer]
Viewer = <Default>
[Options]
Variable Options = 508
Detail = 0
VBPath =
Report Cfg = 60/80/4
User Header =
Last 4 Projects =
WindowSize =
[Control Data Files]
File1 = SCONTROL.DAT
```

[File Viewer]

VBXRef can output the reports it creates directly to a printer or to a disk file. While VBXRef

contains its own file viewer, indicated by the <Default> entry, any Windows program can be used instead.

To use an external viewer, place the name of the executable program in place of the <Default> entry. If the executable program is not in your path or \WINDOWS\SYSTEM directory, you should also provide a complete path.

Example:

```
[File Viewer]
Viewer = \UTILITY\WRITE
```

[Options]

The options heading is followed by seven entries, the first four of which apply to the configuration of your reports:
The "**Variable Options**" and "**Detail**" entries are bit-mapped integers that contain values which correspond to the cross-reference options that were last selected. It is not necessary for you to be concerned with changing these values as VBXRef will automatically update them for you.

"**VBPath = "** is used to specify the fully qualified path *and* filename to your VB.EXE file if it is *not* in your executable path. Since Visual Basic is used to automatically save your project file as text, this option allows you to tell VBXRef the name and location of the Visual Basic executable file.
Example:

```
VBPath = E:\WINDOWS\VB\VB.EXE
```

VBXRef will automatically search your path looking for VB.EXE, but if it has either been renamed or is not in your executable path, you must provide this information in the VBXREF.INI file if you want to use the "Save-Form File(s) As Text" option.

"**Report Cfg**" is the lines per page, characters per line and tab stop settings that are set in the Report Options form.

"**User Header"** is the optional user-defined header that is printed at the top of each page in the Form/Control and cross reference reports. The user header is limited to one line and is entered in the Report Options form. [For more information see Report Configuration.]

"**Last 4 Projects"** is the fully qualified path and filename of the last four Forms and/or Projects opened by VBXRef. The four entries are separated by a CHR$(255) and are automatically updated by VBXRef and listed at the end of the File Menu options.

"**WindowSize"** contains the positioning properties for the main form. VBXRef will save the left, top, width and height properties if the main form is normalized when you unload the program.

[Control Data Files]

These are the names of the control data files that VBXRef will load upon initialization. These files are created by the MhPropc function and are what allow VBXRef to read the form and control information from the Visual Basic .FRM files. If a control data file is not in the same directory as VBXREF.EXE you *must* provide a fully qualified path for the file. If VBXRef cannot find one of the control data files when it loads, an error message will be displayed (giving the name of the file it cannot find) and the program will terminate.

When adding new control data files to the end of the VBXREF.INI file, make sure you use the proper Windows INI conventions: [KeyName] = [Filename]. While the name used in the "KeyName" field does not matter, you ***must*** have some value in that field followed by an equals sign "=" and then the name of the control data file.
Example:

```
File1 = scontrol.dat
File2 = c:\win3\vbxref\vbt1cont.dat
```

For more information on the control data files and how to create them, see *Creating Control Data Files*.

Creating Control Data Files

VBXRef's functionality is broken down into two main areas, which are the two types of reports that it can create:

The Form and Control report, which reads the binary .FRM file and lists all the form and control properties, and

The variable cross-reference report.

The heart and soul of the Form and Control report is the control data file which is created by the MhPropc function. Without a control data file, VBXRef cannot read any of the form or control information. Therefore, it is very important that you understand how to create and maintain these files, because without them, the Form/Control report will not be accurate.

A control data file contains a wealth of information about controls, including class names, properties, events, default values and the bitmaps that appear in the Visual Basic Toolbox window. We provide you with a control data file for the standard VB controls, along with files that describe the custom controls in VBTools (versions 1 and 2). We also provide a mechanism (described later in this section) that you can use to create control data files for other custom controls.

When VBXRef is loaded, it looks for all of the files listed in the [Control Data Files] section of the VBXREF.INI file and then loads all of the data in those files into memory. VBXRef uses several different arrays to store this information, including native VB variable-length string arrays; therefore there is a limit to the number of control data files that you can load. The limit that will usually be reached first is the total number of enumerated properties for all of the controls in all of the data files, which is a maximum of 2000. Should you exceed the maximum capacity of controls or properties, VBXRef will terminate during the initialization process with an appropriate error message. Should this occur, you must remove one or more control data files from your VBXREF.INI file and restart VBXRef.

To create (or re-create) a control data file:

1) Make sure MHUTIL.VBX is in your executable path.

2) Start a new project.

3) File-Add File any custom control VBX.

4) For each control you want listed in the Control Data File, double-click that control's icon to create a single instance of the control. You may reposition the controls anywhere on the form, but you must not change any of the properties!

5) Place the following Declaration in the "General Events" section of your Form:

```
Declare Sub MhPropc Lib "mhutil.vbx" (ByVal filename$, ByVal hwnd%, ByVal
DLLFileName$)
```

6) Place the following line in your Form_Click procedure:

```
MhPropc FileName$, hWnd, DLLFileName$
```

where FileName$ is a variable-length string containing the path and filename of the control data file to create, hWnd is the hWnd property of the Form and DLLFileName$ is the name of the VBX/DLL that contains the controls. We suggest the control data file be given a descriptive name that can be associated with the type of controls in the file. This will make it easier to identify the file should any

of the controls change and the data file needs to be re-created.

The DLLFileName$ parameter allows the MhPropc function to find the bitmap associated with each control and place it in the Control Data File. If you have controls from more than one DLL/VBX on this form, then those that are not in DLLFileName$ will not have a bitmap associated with the controls. When VBXRef reads any controls without a corresponding bitmap in the Control Data File, they will be displayed with a bitmap of a blue question mark next to them.

7) Press F5 to RUN the Project and then click the left mouse button anywhere on the Form (not on a control). This will create the control data file specified in FileName$, which contains all of the information VBXRef will need to read any Form that has one or more instances of the controls.

The DEMPROPC.FRM is an example Form containing the MhPropc declaration and syntax. There are no controls on this Form, so all you need to do to use it is:

1) Load the Form.
2) Load the VBX for which the control data file is to be created.
3) Double-Click on all the controls in the VBX. Do ***not*** change any of the properties.
4) In the Form_Click event, change FileName$ to the filename and path of the control data file you want to create. Change DLLFileName$ to the name of the VBX./DLL you loaded in step 2.
5) Press F5 to run the Form.
6) Click anywhere on the FORM (not on a control). The control data file will be created.
7) End the program.
8) Save the project in case you need to recreate the control data file at a later time.

IMPORTANT: In order for MhPropc to read the bitmaps for the custom controls, you must create a separate control data file for each VBX or DLL. Keeping the controls in separate files also makes updating the control data files much easier whenever you receive an updated VBX for any of your controls. Whenever a custom control VBX/DLL changes, you *must* run MhPropc on the Form containing the controls to insure VBXRef will be able to read the new controls.

The size of each control data file is limited to 60K, which should accommodate most custom control libraries. Should you create a control data File that is greater than 60K, VBXRef will ***not*** be able to load it. Simply remove some of the controls from the form used to create the control data file and place them on a second form, creating two different control data files for this VBX. Be sure you use a different name when creating the second control data file so you do not overwrite the first one.

Now that you have created a control data file for you custom controls, don't forget to add the filename to your VBXREF.INI file!

Resource Usage

There are three main libraries (DLLs) in Windows that contain functions accessible by any Windows program:

"Kernel" provides system services, such as multitasking and memory management,
"GDI" provides the graphics device interface and
"User" provides window management.

The GDI and User libraries use resources: GDI resources and USER resources., each of which are limited to a total of 64K under Windows 3.0 and 3.1. Both types of resources are shared by all of the active Windows applications, including the Windows environment itself. Together, the unused GDI and USER resources are referred to as "Free System Resources", or FSR's.

In most instances where you receive an "Out of Memory" error message from Visual Basic or Windows, the error message is referring to FSR's and not to Windows system memory. The only way to overcome this error is to release some of the FSR's currently in use, which is normally accomplished by unloading forms or terminating active Windows applications.

To minimize use of resources, VBXRef will by default unload all unneeded forms, as well as use several less obvious methods. While there is a slight performance penalty in using these methods, we feel the results are worth it. VBXRef will use approximately eight percent of USER and six percent of GDI resources during normal operation (half of this resource usage is by Visual Basic itself). While the USER percentage will remain static, the amount of GDI usage may increase temporarily as you use certain features, to a maximum of approximately ten percent.

Performance Issues

String operations are the slowest functions in Visual Basic and, as a result, the performance of the variable cross-reference is related directly to the size of your projects and the options you select. For the maximum possible speed, VBXRef uses assembly language routines from our **Muscle** product for all file access and as many string operations as possible. Additionally, VBXRef stores all cross-reference information in memory while it is processing the reports. During processing, none of the data is written to a disk file, even temporarily, so that this operation can be as fast as possible. As a result, the size of the projects that you process is limited by the amount of available Windows memory, including temporary and permanent swap files.

While VBXRef may allocate a large block of memory during the variable cross-reference procedure, this allocation is only temporary and the memory is released immediately after the process is complete. Since most Windows "Out of Memory" errors are a result of low FSR's (Free System Resources) and *not* Windows memory, we felt this was the perfect balance for a program of this nature.

For more information about the variable cross reference module, please see *Cross Reference Reports* under Using VBXRef.

File Operations

Open Project/Form
Save Form File(s) As Text
ViewFile...
Print to...
Printer Setup...
Exit

Open Project/Form - Displays a Windows file-open dialog box that allows you to select the .MAK or .FRM file you wish to load into VBXRef.

Save Form File(s) As Text - This option saves all of the modules of the currently loaded VB project as ASCII files, with the default .TXT extension, in the same directory as the original file. To use this feature, you must have already selected a project or form file from the *"File-Open Project/Form"* selection and Visual Basic must ***not*** be currently loaded. If you load only a single form into VBXRef, this option will be disabled if the form contains any custom controls. This occurs because the custom control DLL filename is saved in the project file and cannot be loaded into Visual Basic.

When you choose this menu selection, VBXRef verifies that all of the files listed in the Project .MAK file exist, loads Visual Basic and minimizes VBXRef. SendKeys statements are then used to load the project or Form file you have selected and to save all appropriate files as ASCII text files. Finally, Visual Basic is unloaded and VBXRef is restored. If VB.EXE is not in your executable path, you must specify a full drive, path and filename in the "VB Path = " line of the VBXREF.INI file.

WARNING: Since VBXRef uses Visual Basic to save the forms and modules as ASCII text files, it does not change the default naming convention of <Filename.TXT>. Therefore, if you have a form and a module that have the same Filename but different file extensions, the module ASCII file will overwrite the form's ASCII file, and VBXRef will think they are one and the same. This option will overwrite any existing files of the same name without confirmation or further prompts.

WARNING: For this option to work correctly, you must be able to load this project without error into Visual Basic. Should VB be unable to load any of the files in the project, the SendKeys statements will fail and the Visual Basic environment will remain loaded. Should this occur, you can either correct the problem and resave the project and exit Visual Basic, or simply close VB and return to VBXRef. The most common reason for this routine to fail is that the project you are attempting to process uses one of the same .VBX files as VBXRef , and Windows has loaded the runtime VBX instead of the design mode VBX. To correct this problem, place the directory for the design mode VBX files ahead of the directory with the runtime files. You should then exit Windows and reload VBXRef.

If you have a form and module that have the same filename but a different file extension, you *must* do the following:

1) Manually save the Form and/or Module file as text using a different filename or extension for one of them.

2) Choose Report Configuration under the Options menu selection and update the name of the text file for the Form/Module. If you do not do this, your cross-reference report will be inaccurate (as well as the Form and Controls report if you choose to include the source code).

View File - Displays a Windows file-open dialog box and passes the selected filename to the file viewer specified in the VBXREF.INI file. If you list an external program which cannot be loaded by Windows, a Message Box appears asking if you want to use the default viewer.

Print To - Displays a dialog box with two options: File and Printer. Whichever one is chosen will be the destination for both Form/Control and Cross Reference reports. VBXRef uses the Printer.Print method to output to the printer, which means the default Windows printer is the output device. (For more information about the Printer.Print method, please refer to your Visual Basic manual or the VB help file.)

Printer Setup - Displays a standard Windows printer selection/setup dialog box. If you make a change to your printer and save the selection, the selected printer becomes the default printer and your WIN.INI is modified. These changes affect any other Windows programs that are currently loaded.

EXIT - Unloads VBXRef.

Reports

There are three items under the Reports Menu selection, but until you load a project or form into VBXRef, the first selection (Form/Control Report) is disabled.

Form/Control Report
Variable Cross Reference
View Report

Form/Control Report - Outputs a listing of all the forms and modules in a project to the selected device. If a single form is loaded, then a report for that individual form is created. If the output device is a file, a dialog box prompts you for the filename to be used; otherwise a message box will prompt you to verify that the default Windows printer is ready.

Variable Cross Reference - Loads the variable cross-reference module. If you have a project loaded, VBXRef will check to make sure it can find a corresponding ASCII file for every Form and Module in the project. If one or more ASCII files are not found, a warning message is displayed. Should you choose to load the cross-reference module after this warning, you will not be able to process the current project, but you can load any existing object data files and generate reports for these files. Object data files (.ODF) are described under *Performance Issues* and *Data File Formats*.

By default, VBXRef looks for ASCII files that have the same filename as the form or module with a .TXT file extension. If you have saved the form or module to a text file using a different name, you should first select "Report Configuration" under the "Options" main menu selection and make sure every form and module has the correct text file associated with it. (For more information, see *"Configuring Your Reports"*.)

If you loaded a project prior to entering the cross-reference module, the "Start" command button is enabled and VBXRef is ready to process the project. You should first press the "Options" command button and select those variables you want to include in your report , as well as indicate if you want Summary or Detail information for each of the selected items.

The "Print Source Listing" check box is visible on the cross-reference form only if a project is loaded into the form viewer and a corresponding .TXT file has been found for every form and module. This option creates an ASCII file containing all of the files in the project, with line numbers (printed in the left column) corresponding to the line numbers listed in the cross reference Summary and Detail reports. The source listing has the same filename as the project, with a .SRC file extension.

View Report - Loads the file viewer specified in the first line of your VBXREF.INI file. If you listed an external program which could not be loaded by Windows, a Message Box appears asking if you want to use the default viewer. If you have generated any reports since loading VBXRef, the name of the last report you created will be the default name displayed in the file-open dialog box.

Options

Report Configuration
Cross-Ref Options
Properties

Report Configuration- The options on this form allow you to customize the output of your reports, as well as select which text files represent the ASCII equivalent of your forms and modules. All of the report configuration parameters in this form are saved in the VBXREF.INI file. Once you set these values, you will rarely need to use this option. For more information see *Configuring Your Reports*.

Cross Ref Options- You can modify the cross reference reports options both from this menu selection as well as by pressing the "Options..." command button on the cross reference report form.

NOTE: In the context of this application, an "object" refers to any variable, constant or literal, in your program. References are the line numbers in the program where the object "appears" in your code. VBXRef generates reports on every object in your program except remarks.

The items listed in the left column are the types of objects that VBXRef can identify and report on. If you check the box for an object type, the "Summary" option button turns on by default. If you uncheck an object, both of the option buttons for that object are unselected.

The type of information listed in the summary report will vary according to the type of object and includes:

Visual Basic Keywords, Numeric and String Literals - The number of references throughout the entire project.

Functions and SubProcedures - Includes the module, scope, line number where the procedure is declared (or located in the source listing) and the total number of references throughout the project. If the procedure is contained in an external VBX or DLL, the module name will be listed as "EXTERNAL". Since external procedures can be declared at either the Global or Module level, it is possible to have an external procedure declared in every Form in the project. External procedures that are module level in scope are listed as "EXT/[Module]", where [Module] is the module name where the procedure is declared.

Line Labels and Numbers - The module, procedure and line number in the source listing where the objects appear and the total number of references.

Constants - The module, scope, line number where the constants are defined and the total number of references throughout the project.

Scalar Variables - The module and procedure (if local) where the variables appear, their scope and total number of references within that scope.

Array Variables- The scope and total number of references within that scope.

Form/Control Properties and Methods- Includes the module where the references occur and the total number of references. It is common to have a number of listings in this category which have zero references because most are usually control procedures that have code "attached" to them, but are never referenced externally in the project.

NOTE: Since VBXRef identifies a user-defined type variable separately from each of the individual elements of that type, it is not unusual to see listings for a user-defined type that has zero references, and directly beneath it, numerous listings for separate elements of the type. The individual listing in the summary will be the line where the typed variable was dimensioned. The dimensioning of the variable itself is not considered a reference to the variable.

Whenever you select the "Detail" option button for any of the listed object types, VBXRef includes the summary information on the first line for each object and then lists the Module, Procedure and Line number of every occurrence in the source listing. When a line number is preceded by an equal sign "=", it indicates that the object was assigned a value on that line. Since it is not possible for VBXRef to tell what variables might return values from external routines, none of the variable names in the invocation of the procedure are considered either a reference or an assignment.

The "Summary ALL" and "Detail ALL" command buttons allow you to select the detail or summary options for all of the object types at once.

If you loaded a project prior to displaying the cross-reference options form, the "Adjust Obj/Refs" command button is enabled. Pressing this button displays a separate form that shows you the following information:

"Available Memory " is the total amount of Windows memory that will be available *after* VBXRef dimensions the objects and references arrays with the values shown. This value includes the amount of memory available in the temporary or permanent swap file (if one exists).

"Objects" and "References" represents VBXRef's estimate of how many objects and references will be required for your project, based on the total size of all the forms and modules in your project.

If you attempt to create a cross-reference report that runs out of available objects or references, you should increase the number of objects and/or references and run the report again by pressing the "Start" command button. You can modify the number of objects and references by entering the number directly into the field or you can use the spin controls (the up and down arrows). The spin control for the objects changes the value in increments of 10, while references are changed in increments of 25.

Performance Issues - VBXRef stores all objects and references in memory for maximum speed. While VBXRef has the capability to report on all Visual Basic Keywords, as well as numeric and string literals, we suggest you do not choose these options unless you absolutely need them. Checking these three options will increase the processing time by 50 to 100 percent and more than doubles the memory requirements for the variable cross referencing!

When VBXRef finishes processing the project, your report is ready to be output to the selected device. If the printer is your output device, a dialog box prompts you to verify your printer is on line and ready. If a file is your selected device, a File-Open dialog is displayed with a default name of [ProjectName].VXR supplied.

After the report is created, VBXRef asks if you want to save the object and reference data files. If you answer YES, the files are saved using the project filename and the following file extensions:

.ODF for the objects data file
.RDF for the references data file
.SDF for the string literals, if this option was selected.

Since there can be three different data files associated with a project, the file extensions for these files are *not* modifiable. If you have previously saved the data files for this project, you will be prompted before the existing files are overwritten.

Once the objects and references are saved, you can load them at a future time, change the options if needed and recreate your reports without waiting for VBXRef to read all of the source code in your program. To load existing data files into VBXRef and create reports, press the command button labeled "Load Data Files". A "File Open" dialog box prompts you to select an object data file (.ODF). Once you select a valid object data file, all of the corresponding files will be loaded and a new cross-reference report is created according to the options that are selected.

NOTE: If you modify any module or form in your project and want the changes to be reflected in your cross-reference report, you must process the entire project again.

NOTE: If you did not check Visual Basic Keywords or either of the literal options when you processed the project and saved the data files, you will not be able to add them to your reports at a later date without loading and reprocessing the project. All of the other objects are saved in the data files, regardless of the options selected when the project was processed.

If you start VBXRef with a command line parameter of "PARSE", the cross-reference module is loaded (instead of the Form Reader). In this case, there is no project loaded into VBXRef and you can create reports only from existing object data files by using the "Load Data Files" command button.

Properties - Selecting this option displays a sub-menu with three selections:

Non-Default Only - Checked by default. VBXRef includes only those properties that are different from their default values.

Exclude Positioning Properties - Checked by default. VBXRef excludes the positioning properties [Left, Top, Height, Width]. Since there is never a known default for these properties, they always appear as Non-Default properties. Since most programmers don't normally have a need for this information, this option allows you to exclude the positioning properties from your Form Report.

Show/Print All Properties - Selecting this option disables the first two selections and all control properties are included in the report.

Help

Index- Starts WINHELP.EXE (if needed) and loads the VBXREF.HLP file. This help file *must* be either in the same directory as VBXREF.EXE., your \WINDOWS or \WINDOWS\SYSTEM subdirectories or anywhere in your executable path.

About- Lists the version number and tells you who created VBXRef.

General Information

While Visual Basic is an excellent Windows development environment, like many 1.0 versions, there are some areas that could use some help or improvement. One is the management of your source code, which is exactly what VBXRef is designed to help you with.

Visual Basic stores all of the data for a project in four different types of files:

> The *project* (.MAK) file is a binary file that contains the name of all forms, modules and custom control DLL's used in your project. This file also indicates which form is the "Start-up form", the project title (if any), as well as the position and state of the project window the last time this file was saved. All filenames in the project file are stored with a path relative to the .MAK file itself.
>
> The *global module*, which is always an ASCII text file.
>
> *Form* files are always in binary format and contain your control and form information and any source code saved as threaded p-code.
>
> *Module* files can contain external routine definitions as well as global subprocedures and functions. By default, module files are saved in binary format with the source code as threaded p-code. You can also save these modules as text, using the "Code-Save Text.." option in VB. If you save a module as text and replace the binary file in the project with the ASCII file, VB will continue to save the module in text format. The only way to resave these files in binary format is to create a new module and then cut and paste the code from the ASCII file into the new module. One reason you may want to keep your modules in binary format is that both VB and VBXRef can load these files faster than their ASCII equivalents.

While VB can also save your .FRM files as text, the only data included in these ASCII files is the code in the "General" section of the form, all control procedures that have code in them and any local procedures you have created. While the code in the General section is always at the top of the file, the other procedures are output in a random pattern and are not sorted. The point of all this is that Visual Basic does not save any of the control or form information in the text files. Since the forms and controls are an integral part of most VB applications, this can make management of your source code extremely difficult.

VBXRef can read the binary format of the Visual Basic form and module files, allowing you to create reports that contain all of the information about your project. It also groups all of the controls of the same type and optionally includes the source code attached to the active procedures, which it sorts and prints in alphabetical order. If you want VBXRef to include the source code in the Form/Control report, you must first save your forms and modules "As Text" (VBXRef can do that for you automatically).

Custom Controls

VBXRef can read all the property information for any form, all standard controls and *most* custom controls. However, since Visual Basic allows the custom control programmer to write their data to the .FRM file in most any format, there are custom controls that write their property information to the .FRM file in a method that VBXRef cannot read, even with the control data file as its guide. The most common cause of this problem is controls that use bit-mapped integers and store the values for more than one property in the same byte(s).

NOTE: This does not mean the custom control programmer has done anything incorrect or improper, since prior to VBXRef, the only thing that needed to read this information was the custom control itself. In order to remedy this situation, MicroHelp is developing a way to provide companies that write custom controls the ability to provide you with a special data file that enables VBXRef to read their data in the .FRM file. If you have a custom control library that creates .FRM files that VBXRef cannot read, please ask the publisher to contact MicroHelp and we will be happy to provide them with the specifications for this data file.

Whenever VBXRef encounters data in the form file that it cannot understand, the program displays an error message number 100. This error indicates one of:

The form contains custom controls that have written to the file in a format that VBXRef cannot understand, or

The control data file that you loaded for this custom control is older than the DLL and the DLL contains some new or changed properties. You must re-create the control data file and restart VBXRef. Please refer to *Control Data Files* for details.

Even if VBXRef cannot read all of the property information for a custom control, it should always be able to correctly identify the type of control, display the correct bitmap for the control in the form viewer and display the control names of all instances of that control on the form.

Loading a Project

From the VBXRef Menu, select *File - Open* and choose a VB project or form from the dialog box. If you select a project (.MAK file), all of the file names in the project are displayed in the left list box, just as they appear in the Visual Basic "project" window. Single click (or move the highlight bar to) any of the .FRM files and all of the form information is placed in the right list box. If you select a module file, the "Properties" window is displayed and all procedures in the module are shown.

If you selected a single form file from the file-open dialog, VBXRef immediately loads the information for that form into the right list box.

The first two entries in the right list box are *always* *General Events"* and *"Form"*. If there are any general procedures in the Form, a green check mark is displayed to the left of the *"General Events"*.

If you double-click the *"General Events"* item in the right list box, the "Property Listing" window is loaded. This window contains two list boxes with the names of all general procedures for the form displayed in the right list box. If you double-click on the *"Form"* item, the form's properties values and active events are displayed.

If there are any controls on the form, they are grouped by stylename and listed alphabetically. When you double-click on a stylename, the list box expands, showing you the name of all the controls of that type (stylename) that are on the form. If you then double-click on a control name, the property names, values and any active events for that control are displayed.

The number and type of properties displayed are controlled by the *Properties* Menu Selection under *Options* on the VBXRef Main Menu. If *"Non-Defaults Only"* is checked, only those properties that are different from the default values are displayed. If *"Exclude Positioning Props"* is checked, the Height, Width, Left and Top properties are excluded. If you want all of the control properties displayed, choose the "*Show/Print All Properties"* menu selection. These selections are also honored when generating the form and control report.

Double-clicking on an "expanded" control type in the right list box causes the list box to contract, and the control names will no longer be displayed for that stylename.

All properties that have been changed from their default settings are displayed in *red*. You may notice that all of the properties that are variable-length strings (Caption, CtlName, Text, etc.) are *always* displayed as non-default . This is because string properties do not have a *default* value stored in the control data file. In addition, the positioning properties are always considered non-default.

If there are any resources (.BMP, .ICO or .WMF images) stored in the form file, you can display the image by double-clicking on the corresponding property name. This action causes the "Resource Viewer" window to be displayed. If the image is larger than the window, you can scroll the image with the horizontal and vertical scroll bars.

You can close the "Resource Viewer" as well as the "Property Listing" window with a right mouse click anywhere in the window or by double-clicking the control box. If you resize the Property Listing, these sizes will remain in effect throughout the current session of VBXRef or until you resize the window.

Configuring Your Reports

The *Report Configuration* option under the "Reports" menu selection displays a window that allows you to specify the number of printable lines per page, the number of characters per line, the tab stop setting, a check box to exclude page breaks and whether VBXRef should include the source code for the active events in your report.

In order to include the source code in the form and control report, you must first save the form files as text. (Use the Code, Save As Text option (Alt-C, S) from the VB menu or you can let VBXRef save all of your forms and modules as text using the *File-Save Form File(s) As Text* menu option.)

While it is recommended that you save the .TXT (text) files in the same directory as the .FRM (form) files, you are not required to do so. If the ASCII files are saved in a different directory from the form and module files, or if you use a file extension other than .TXT, you will have to use the "Report Configuration" option under the "Reports" menu selection to tell VBXRef which text file corresponds to which form/module. To accomplish this, press the command button labeled *"Select/Edit Text Files"* and two list boxes will be displayed. All of the form and module files in the current project will be displayed in the left list box. The text files that VBXRef has found are displayed in the right list box. (If a corresponding .TXT file cannot be found, a "<Not Found>" entry will be placed in the corresponding position in the right list box.) If you have saved any of the <Not Found> files as text using either a different filename, file extension or directory, highlight the selection you want to change and press the "Add/Change" command button to update the right list box.

If you check the *"Print Source"* check box on the Report Configuration Form, but the corresponding ASCII text file cannot be found, VBXRef will still print the report, but the program will just list the names of the active events in the same manner as if this option were not chosen.

Form/Control Reports

The Form/Control Reports are what makes VBXRef very unique and provides information that was never before available to the Visual Basic programmer.

When creating these reports, VBXRef uses a default file extension of ".REF", but you are welcome to use whatever extension you wish.

The following is a sample Form/Control report that uses the default settings, listing only non-default properties and no source code. The page length is set to 60 lines, the column width to 80, tab stops to 4 and page numbering and headers are enabled.

```
Page 1    VBXRef Form/Control Report 04-10-1992   23:09:54
Test user header
Project: E:\WIN3\VB\XREF2\VBXREF.MAK

File Summary:
StartUp Form: Sub Main
       E:\WIN3\VB\XREF2\GLOBAL.BAS 04-08-1992  7:04:38 am
       E:\WIN3\VB\XREF2\ABOUT.FRM 04-05-1992  9:16:12 am
       E:\WIN3\VB\XREF2\BMPFORM.FRM 04-04-1992  9:45:00 pm

Form: ABOUT.FRM
       General Procedures:
              TestProc

              BackColor     &HFFFFFF
              BorderStyle   3 - Fixed Double
              Caption       VBXRef
              ForeColor     &H0
              FormName      About
              LinkTopic     Form2
              MaxButton     False
              MinButton     False

              Active Form Event Procedures
                Form_Click   Form_Load   Form_Unload

       Control Type: PictureBox
              Control Name: AboutPicture
              BorderStyle   0 - None
              CtlName       AboutPicture
              Enabled       False
              Picture       (Icon)
              TabIndex      1
              TabStop       False

              Active Control Event Procedures
                <None>
```

While all of these items should look familiar and be self-explanatory, please note that on the first page, under "File Summary:", VBXRef lists the name of the startup form for the project, which in this case is the "Main" subprocedure.

The information for the forms and modules is listed in the same order as the files appear in the VB

project window (forms first, followed by modules, each sorted alphabetically). For each form, the report lists the name of all active "general" subprocedures and functions, followed by the property values for the form itself. If there are any active form event procedures, the names will be listed.

Finally, the report lists all of the controls (grouped by their stylename), the control properties and active control event procedures.

The standard controls are not sorted alphabetically by their stylenames, but instead appear in the same order as the Visual Basic Toolbox window: Picture box, Label, Text Box, etc.

All standard controls are listed first, followed by any Menus and then the custom controls (if any). The order of custom controls is the same order in which they were loaded in your VBXREF.INI file.

If you saved each of your forms and modules "As Text" and selected the check box on the "Report Configuration" form to include the source code with the Form/Control report, the report might appear like the one shown below. This option allows you to combine your source code with your form and control information into one report. Prior to VBXRef, this type of report was not possible.

```
Page 1    VBXRef Form/Control Report 04-10-1992   23:09:54
Test user header
Project: E:\WIN3\VB\XREF2\VBXREF.MAK

File Summary:
StartUp Form: Sub Main
  E:\WIN3\VB\XREF2\GLOBAL.BAS 04-08-1992  7:04:38 am
  E:\WIN3\VB\XREF2\ABOUT.FRM 04-05-1992  9:16:12 am
  E:\WIN3\VB\XREF2\BMPFORM.FRM 04-04-1992  9:45:00 pm

Form: ABOUT.FRM
  General Procedures:

    Static Function TestProc (One)
        'this is a do nothing proc
    End Function

    BackColor     &HFFFFFF
    BorderStyle   3 - Fixed Double
    Caption       VBXRef
    ForeColor     &H0
    FormName      About
    LinkTopic     Form2
    MaxButton     False
    MinButton     False

    Active Form Event Procedures
    Sub Form_Click ()
        Show SuperPic
    End Sub

    Sub Form_Load ()
       Move (Screen.Width-Width) \ 2,(Screen.Height - Height) \ 2
        Mh3D1.Caption = "Visual Basic Cross Reference
    ⇒     Utility" + Chr$(13) + "Copyright 1992 -
    ⇒     MicroHelp Inc. All Rights Reserved."
```

```
  End Sub

  Sub Form_Unload ()
      TerminateProg True
  End Sub

Control Type: PictureBox
  Control Name: AboutPicture
  BorderStyle  0 - None
  CtlName      AboutPicture
  Enabled      False
  Picture      (Icon)
  TabIndex     1
  TabStop      False

  Active Control Event Procedures
    <None>
```

Variable Scoping

The "scope" of a variable determines which parts of the project can "see" or access the variable, which is why the scope is also commonly referred to as "visibility". Since VB allows variable names to be used only for a single type within the same scope, you should keep this principle in mind when creating your variable names.

The method we often use is to start all Global Constants with a "gc", followed by a descriptive variable name. For Global variables we use just the lower case "g" and for module level variables we use a lower case "s" (this is a carryover from SHARED variables in Microsoft QuickBASIC).

```
Examples:
       Global Const gcTrue = -1
       Global gMyVariable
       Dim sFoo As String * 10
```

By using this method, you can tell the scope of variable simply by its name. This can save you a lot of trouble when you're debugging a program and you don't understand why the value of a variable seems to change unexpectedly, or Visual Basic gives you a "Type Mismatch" or "Duplicate Definition" error message.

In Visual Basic, there are three different levels of scope or visibility.

Global. All global level variables and declarations are visible to all forms and modules in the project. Only three things can be declared or defined *only* in the global module: Global Constants, Global variables and user-defined type definitions. While many programmers declare external subprocedures and functions in the global module, in large projects this may cause a problem due to the module's 64K limit. Should you be approaching this limit, or you want to modularize your code, you may want to consider placing external declarations in modules instead of in the global.

Form or Module. All variables that are Dim'ed or Dim Shared in the "General" section of a form or module are visible to all subprocedures and functions throughout that form or module.

Local. Variables whose values are visible only within the subprocedure or function in which they are used are considered local in scope. These variables are initialized to zero (or null, in the case of strings) upon entry to the procedure and they are erased whenever program execution exits the procedure in which they are used.

NOTE: "Static" variables, as well as all locally scoped variables within a static procedure, are not initialized upon entry to, or erased upon exit from a procedure. On the contrary, the last value in the static variable persists until you either change the value of the variable or the program terminates. Values placed in static variables will even persist inside forms that are unloaded. If you use the Static keyword, you need to be aware of this "feature" so you will know to initialize these variables when necessary.

Not only do variables have scope in Visual Basic, but so do subprocedures and functions (which we refer to collectively as "procedures"). However, there are only two levels of scope for procedures in VB: "global" and "form" (or "module") level.

The only procedures you can declare in your global module are external (those that reside in a DLL/VBX). For example, all Windows API services are external procedures.

As we mentioned earlier, you are not required to declare your external procedures in your global module to make them global in scope. All external procedures that are declared in the "General" section

of a *module* (not a form), as well as all Visual Basic module procedures (not form procedures), are global in scope and thereby accessible throughout your entire project.

HINT: When a Visual Basic application is first loaded, all code that resides inside modules is loaded before any forms. For large projects, there can be a significant delay between the time the program is executed and when the first form appears on the screen. If your startup form is large (either has a lot of code and/or controls), you may want to show a small "Initialization" form while your startup code is being loaded. To do this, make the "Sub Main" procedure your startup form and inside this procedure Show your startup form (causing it to load). All module level code remains in memory throughout the life of your application and is not unloaded until your program terminates. Since modules consume only memory (not FSR's), you should place all code that is not unique to any one form inside a module for use throughout your entire application.

All external procedures declared in the "General" section of a form, as well as those that are a part of a form, are visible only inside that form. While you can declare external procedures at the form level, it is generally not efficient to do so.

VBXRef will tell you the scope of every function, subprocedure and variable within your entire project as well as how many times that each item is referenced.

Many times VB programmers will cut and paste into the global module a large group of Windows API declarations and global constants that are never used. Since these declarations both increase the size of your .EXE and count toward the 64K limit, VBXRef can tell you which routines you are not using and you can then remove from your code. Our GETDEC utility that is included with VBXRef will read your source code and create a listing of only those declarations used in your program. Please refer to the GETDEC.HLP file for more information.

Cross Reference Report

When the cross-reference window is loaded, the main VBXRef window is minimized and cannot be accessed until you close the cross-reference form. In order to generate a cross reference report, you *must* first save each and every form and module in your project "As Text", using the "Code, Save As Text" option from Visual Basic, or have VBXRef do this for you by using the "Save Form File(s) As Text" selection from the File menu.

If a .TXT file cannot be found for each form and module in the project, a warning message box is displayed. In this situation, you can still choose to load the cross reference module, but the only option available is to load an existing object data file. For more information on determining which ASCII files could not be found, please see *Configuring Your Reports*.

When you first load the cross reference form, you should make sure that the desired options are set by clicking the "Options" command button. Make any desired changes and then "Close" this form.

If you loaded a project into VBXRef prior to selecting the variable cross-reference module and all of the corresponding .TXT files were found, the "Start" command button is enabled and the label at the top of the form indicates that VBXRef is ready to process the project. When you click the "Start" command button, VBXRef determines the size of every ASCII file in the project and allocates the arrays for the data files based on the total file size and the reporting options you have selected.

If you do not have enough memory available (based on these calculations), VBXRef will display a warning in a message box, and ask if you wish to continue. If you answer "YES", VBXRef will begin processing your project. If you answer "NO", VBXRef will return you to the cross-reference window. You can either close this window and return to the main VBXRef window, or manually adjust the number of objects and references you want VBXRef to allocate before processing the project. If VBXRef reaches the limits of either the objects or references arrays, an error message is displayed and the cross-reference process is aborted.

To manually adjust the number of objects and references, click the "Options..." command button to display the cross-reference options window and then click the "Adjust Objs/Refs" command button. A small modal window is loaded that shows the current allocation of objects and references for your project. You can enter new figures for either field or use the spin controls to adjust the numbers up or down. As you change the number of objects and references, the amount of free Windows memory will be updated at the top of the form. VBXRef allows you to select up to 999,999 objects and references, but only up to the limit of available Windows memory. In most cases, the numbers estimated by VBXRef should be sufficient to complete your project and we do not suggest that you increase these numbers unless VBXRef aborts during the cross reference processing.

When you press the "Start" command button, VBXRef examines each ASCII file in the project. The labels on the cross-reference window keep you updated as to what file is being processed and the percentage of completion. The two labels on the bottom of the form show you what percentage of the allocated objects and references arrays have been filled, as well as the total number of objects and references that have been found.

When VBXRef finishes processing the project, your report is ready to be output to the selected device. If the printer is your output device, a dialog box prompts you to verify your printer is on line and ready. If a file is your selected device, a File-Open dialog is displayed with a default name of [ProjectName].VXR supplied.

You can cancel both the cross reference processing and report output at anytime either by pressing the <Esc> key or by clicking the "Cancel" command button. If you choose to cancel processing, a message box prompts you to verify cancellation before the processing or the report is aborted.

Once the report is finished, a message box is displayed asking if you want VBXRef to save the data files that were created for this project and used to create the report. If you answer "YES", the data files are saved using the filenames specified in the next section. All of the memory allocated for the data files is released and you are returned to the variable cross reference window.

Data file formats

After you create a cross reference report, you have the option to save the data used in creating the report. This allows you to load the data at a later date and reprint the report with the same or even different options. If you tell VBXRef to save the data files, a minimum of two and a maximum of three binary data files are created in the same directory as your .MAK or project file. These files have the same filename as the project file and use the following file extensions:

.ODF - Object Data File: This file contains the name of all the objects, the number of times each is referenced and the names of all the forms and modules in the project. The data is sorted by the first two fields (Object Type and Name). The object data file record format is shown below, with the numbers indicating the offset of each element in the type. The file formats are provided in the event you want to create a customized report using this data.

```
Type  ObjDataFile
      Type As Integer      '0 Object Type (see below)
      Name As String * 40 '2   Object Name
      Number As Long       '42 Object Number
      Scope As Integer     '46 Variable Scope (see below)
      Dummy1 As Integer    '48 Not currently used
      DefInMod As Integer '50 Defined in module number
      InProcNo As Long     '52 Defined in procedure number
      OnLineNo As Integer '56 Defined on line number
      NumRefs As Integer   '58 No of ref's to this obj
      Dummy2 As Long       '60 Not currently used
End Type
```

The two DummyX elements were added to fill the Type structure out to a total of 64 bytes. In order to use the **Muscle** array routines to save and load these files with one function call, the record length must be a power of two.

The object types are one of the following values shown below:

```
Global Const gcVBKeyword =    0 'VB Keywords
Global Const gcNumLits =      1 'Numeric Literals
Global Const gcFunc =         2 'Functions
Global Const gcSub =          3 'Subprocedures
Global Const gcLabel =        4 'Line labels and numbers
Global Const gcConstant =     5 'Constants
Global Const gcScalar =       6 'Scalar Variables
Global Const gcArray =        7 'Array Variables
Global Const gcControlProps = 8 'Form/Control properties
Global Const gcLitStrings =   9 'Literal strings
```

The .Scope element contains one of the following values:

```
Global Const gcLocalLevel =  0
Global Const gcModuleLevel = 1
Global Const gcGlobalLevel = 2
```

.RDF - Reference Data File: This file contains information about each reference to every object in the object data file. This includes the line number, procedure number and module where the reference occurred. If the object is a variable, this will also indicate if the value of the variable was changed on this line. The Reference data file record format is shown below:

```
Type ReferenceInfo
     ObjNo As Long       ' 0  Object number
     Module As String * 1       ' 4  Used in Module
     InProcNo As Long           ' 5   in Procedure number
     LineNo As Long             ' 9   on Line number
     DummyStr As String * 1     '13  not used
     Dummy As Integer           '14  not used
End Type
```

The two Dummyx elements were added to fill the Type structure out to a total of 16 bytes. In order to use the **Muscle** array routines to save and load these files with one function call, the record length must be a power of two.

.SDF - String Data File: If you selected "String Literals" when you created the cross reference report, all of the string literals for the project are stored in this file, sorted in alphabetical order. The first four bytes of each string is a long integer (binary format) indicating the object number which relates it to the other files.

The .SDF file is not a random access file, but rather a dump of memory from a **Muscle** Huge String Array. To read this file you will need the function MhHugeStringRead in MicroHelp **Muscle** for Visual Basic.

If you have MicroHelp **Muscle** for Visual Basic, we have included the VB source code used to load the data from the .ODF and .RDF files in the READFILE.SUB file. The MhHugeStringRead function is also a part of Muscle and can be used without modification. Please make note of how the form and module names are stored in the first records of the object data file. If you do not have **Muscle**, you can still access the data in the .ODF and .RDF files using the VB Get function, but we don't provide the procedures to do this.

CRITICAL Errors:

An error so severe has occurred that VBXRef can no longer continue operation. After you press the "OK" Command Button on the error notification message, VBXRef will close all open files, release all resources and terminate. All critical errors are numbered from 1 to 99.

Error 1: VBXRef could not find your VBXREF.INI file, so it attempted to create a default INI file and while attempting to write that file to your hard disk, an error occurred.

> *Recommended Action:* Exit Windows and check your available disk space. You may also want to do a CHKDSK or run some other diagnostic program to check for lost or cross linked clusters. If you have sufficient disk space and there are no hardware problems with your P.C. then delete the VBXREF.INI file and restart VBXRef. See *The VBXREF.INI File* for more information on the INI file format.

Error 2: VBXRef was unable to read your VBXREF.INI file.

> *Recommended Action:* Delete the VBXREF.INI file and restart VBXRef. See *The VBXREF.INI File* for more information on the INI file format. This error would normally only occur if you edited the VBXREF.INI file and modified it in such a way that VBXRef could no longer read it.

Error 3: An error occurred while attempting to sort a Huge String Array containing the information in the Control Data files.

> *Recommended Action:* This error should only occur if you are low on memory or system resources. Close any other active Windows applications and re-load VBXRef.

Error 4: The file listed in the Error Message Box is not a valid VBXRef control data file.

> *Recommended Action:* Verify that all of the control data files listed in your VBXREF.INI file were created with the MHUTIL.VBX. See *Creating Control Data Files* for more information.

Error 5: VBXRef was unable to DIMension the Huge String Array needed to store the information in the Control Data files.

> *Recommended Action:* This error should only occur if you are low on memory or system resources. Close any other active Windows applications and re-load VBXRef.

Error 6: The control data file listed in the message box is greater than 60,000 bytes in size. While VBXRef can load multiple control data files, the maximum size for any one file is 60,000 bytes.

> *Recommended Action:* Find the control data file that is too large, remove some of the controls and then re-create the file.

Error 7: An error occurred while loading the control data file into memory. For maximum speed, VBXRef loads an entire file into memory at one time and then formats the data and stores it into a Huge

String Array.

Recommended Action: This error should only occur if you are low on memory or system resources. Close any other active Windows to increase the amount of available memory and restart VBXRef.

Error 8: VBXRef was unable to store the data from a Control Data file into a Huge String Array.

Recommended Action: By the time VBXRef is storing the data into the Huge String Array, all the memory it needs for this task has been allocated. Before unloading VBXRef by clicking on the OK button in the Error Message Box, activate Program Manager and check your available memory and resources. Close any other active Windows and restart VBXRef.

Error 9: Similar to error number 8. VBXRef had successfully stored all of the information from your Control Data Files into the Huge String Array and was attempting to store some default values into this array before sorting.

Recommended Action: By the time VBXRef is storing the data into the Huge String Array, all the memory it needs for this task has been allocated. Before unloading VBXRef by clicking on the OK button in the Error Message Box, activate Program Manager and check your available memory and resources. Close any other active Windows and restart VBXRef.

Error 10: You have exceeded the maximum number of allowable Controls and Properties that VBXRef can store from the Control Data Files.

Recommended Action: Either decrease the number of Control Data Files listed in your VBXREF.INI file or the number of Controls in one or more of the data files.

Error 11: You have exceeded the maximum number of allowable enumerated list entries that VBXRef can store from the Control Data Files. An enumerated list are those entries which are selected from the Properties Bar for an individual property of any control.

Recommended Action: Either decrease the number of Control Data Files listed in your VBXREF.INI file or the number of Controls in one or more of the data files.

Errors 12-99: Reserved for future use.

WARNING Errors:

An error has occurred that will effect the accuracy and/or efficiency of VBXRef but is not so severe that the program must abort. All warning error messages are in the range of 100 to 199.

Error 100: This is the most common error you will receive with VBXRef. It occurs when VBXRef is unable to correctly read the properties for a custom control from the Form file and is caused by one of the following: 1) The custom control uses a "non-standard" method to write property information to the Form file. 2) The custom control has been changed since the Control Data file was created. Compare the date of the VBX or DLL file to the date of the Control Data file. While this is not a critical error, some of the information for this control will be *not* be correct.

> *Recommended Action:* Re-create the control data file for this control. If the control has not been modified since the control data file was created, then the control is using a unique method to write its information to the Form file. Please see the section on contacting *Technical Support* for more information on forwarding an example form that duplicates the problem.

Error 101: VBXRef was unable to locate the stylename for this custom control. The stylenames, which are displayed in the right list box, are usually (but not always) the same as the name of the custom control. The normal cause of this error is this control is not in any of the control data File(s) that are in your VBXREF.INI file.

> *Recommended Action:* Verify the data for this control is in one of the control data files listed in your VBXREF.INI file. If necessary, re-create the control data file for this control. If this does not resolve the problem, please see the section on contacting *Technical Support* for more information on forwarding an example form that duplicates the problem.

Error 102: VBXRef was unable to find a property for this control in one of its internal arrays. This error is very rare, but if it occurs some of the information for this control will be incorrect.

> *Recommended Action:* Re-create the control data file for this control. If this does not resolve the problem, please see the section on contacting *Technical Support* for more information on forwarding an example form that duplicates the problem.

Error 103: While attempting to read the Menu properties for this control, VBXRef found a byte it did not recognize.

> *Recommended Action:* Load this form into Visual Basic and make sure it has not been corrupted. Make sure you examine the Menu values and then resave the Form and try VBXRef again. If this does not resolve the problem, please see the section on contacting *Technical Support* for more information on forwarding an example form that duplicates the problem.

Error 104: VBXRef was unable to open or read the Project .MAK file.

> *Recommended Action:* If you are on a network, make sure no other workstation has this file open and the .MAK file has not been damaged.

Error 105: An error occurred while attempting to sort the controls in this form.

Recommended Action: Make sure the form file has not been damaged by loading it into Visual Basic, and running it in the environment. Resave the form and try VBXRef again. If this error occurs again, please see the section on contacting *Technical Support* for more information on forwarding an example form that duplicates the problem.

Error 106: A memory error occurred while loading the form file into a buffer for parsing.

Recommended Action: Check the amount of available memory and system resources. Unload any other Windows applications and try again.

Error 107: You have exceeded the maximum number of objects allocated for this project. All of the objects and references are stored in "Huge" arrays that are dimensioned when the processing started. Since there is no way to increase the size of these arrays and maintain the data stored within, the processing must be aborted.

Recommended Action: Use the "Cross-Reference Options" form to increase the maximum number of objects and start the process again. (See the section on *Cross Reference Reports* for more information.)

Error 108: You have exceeded the maximum number of References allocated for this project. All of the objects and references are stored in "Huge" arrays that are dimensioned when the processing started. Since there is no way to increase the size of these arrays and maintain the data stored within, the processing must be aborted.

Recommended Action: Use the "Cross-Reference Options" form to increase the maximum number of References and start the process again. (See the section on *Cross Reference Reports* for more information.)

Error 109: VBXRef was unsuccessful in DIMensioning one of the arrays necessary to store the cross reference information. Windows shows sufficient memory, but refused our request for the allocation. This normally occurs when you are running Windows in Standard mode and the allocation is 70% or more of the available memory.

Recommended Action: Use the "Cross-Reference Options" form to decrease the maximum number of Objects and/or References and start the process again. (See the section on *Cross Reference Reports* for more information.) If you are running Windows in Standard mode, restart Windows in Enhanced mode and use either a permanent or temporary swap file.

Errors 110 - 199: Reserved for future use.

INFORMATIONAL Errors:

A minor error occurred that does not effect the accuracy or efficiency of VBXRef, but is provided to the user for possible corrective action. All informational error numbers have a error number of 200 or greater.

Errors 200+: Reserved for future use.

.FRM Definition

Visual Basic stores all of the information for a Form in a binary file with a default .FRM extension. This file includes all of the information about the controls, including their properties and values as well as your source code, which is saved as threaded P-code. When you use the "Save-As-Text" option, VB will create an ASCII text file that contains the source code in the General Event Procedures, but nothing about the Controls or the Form itself are written to this file. VBXRef is the only program that can read the Control and property information from the .FRM file.