# Don Malin's
# Cross Reference Program

by Crescent Software

# Don Malin's Cross Reference Program

Program and documentation by Don Malin

TABLE OF CONTENTS

# Requirements

## REQUIREMENTS

This cross reference program is intended for use on an IBM PC, XT, AT, PS2 or 100% compatible computer running DOS 3.0 or later. At least 512K of memory is required, however 640K of DOS memory and at least 64k of expanded memory is recommended for optimum performance and data space.

The program can track up to 2,425 program objects (procedures, labels, constants, and variables) and up to 32,766 separate references if 640K of memory is available. If your system has less memory, the program will be able to handle fewer objects and references.

In order to leave as much memory as possible for object tables, the program uses code overlays. If you have at least 64k of expanded memory, the program will swap overlays from memory. If EMS memory is not available, overlaid code is swapped from disk thus slowing it's operation somewhat.

If you intend to modify the source code for the XREF.EXE program and recompile it, you will need Microsoft QuickBASIC 4.00b or later (BASIC 7, PDS is recommended) and Crescent Software's QuickPak Professional version 3.0 or later.

# 1. INTRODUCTION

Welcome to Don Malin's Cross-Reference program (XREF). We have made every effort to make this product the most comprehensive source code analysis tool available for BASIC programmers. You can now track program elements (objects) over an entire multi-module program, find objects that are defined but never used, and create formatted reports to fully document your program. If you have ever had to modify someone else's program or a program you haven't worked on in a long time, you will appreciate the clarity of these reports.

We sincerely hope that you find XREF both useful and informative. If you have a comment, a complaint, or perhaps a suggestion for another QuickBASIC-related product, please let us know. We want to be your *favorite* software company.

Before we begin discussing the contents of the XREF disk and manual, please take a few moments to fill out the enclosed registration card. Doing this entitles you to free technical support by phone, as well as ensuring that you are notified of possible enhancements and new products. Many upgrades are offered at little or no cost, but we can't tell you about them unless we know who you are! Note, however, that if you purchased XREF directly from us, the mail-in portion of the registration card may have been removed. In this case, you are already registered. We have done this both for your convenience and for ours.

Also, please mark the XREF product serial number on your disk label or manual cover. License agreements and registration forms have an irritating way of becoming lost, and doing this will insure that the number is handy if you need to contact us. You may also want to note the product version number in a convenient location; this is stored on the distribution disk in the volume label. If you ever have occasion to call us for assistance, we will need to know your serial number, and probably the version you are using as well.

To determine the version number for any Crescent Software product, simply use the DOS VOL command, which will display the disk volume label:

```
VOL A:
Volume in drive A is XREF V1.00
```

We are constantly improving all of our products, and you may want to call us periodically and ask for the current version number. Major upgrades are always announced, however minor additions or fixes are generally not. If you are having any problems at all -- even if you are sure it is not caused by one of our products -- please call us. We support all versions of QuickBASIC, and can often provide better assistance than Microsoft.


## WHAT THE PROGRAM DOES


XREF uses pulldown menus and dialog boxes throughout, to make it's operation as easy and intuitive as possible, while allowing the user to tailor it's functionality.

The program and this documentation often refers to program *objects*. This term refers to any distinct program element such as BASIC keywords, procedure names, line labels, constants, and variables.

The XREF.EXE program can be used to read and analyze any Microsoft BASIC (BASICA, GWBASIC, or QuickBASIC) program. Several report types can be generated such as formatted source listings and sorted lists of program elements and their usages. All reports can be sent to either a printer or a disk file. Unlike other cross-reference utilities you may have seen that merely list your variables, XREF provides a *complete* report on all aspects of your program.

Source listings may be generated with headers and line numbers for reference with other reports. If specified, procedures can also be printed on separate pages, and their page numbers will be listed in the table of contents that is automatically generated for all reports. Listings will include all modules listed in your program's .MAK file including all '$INCLUDE files encountered.

Quoted strings and comments may also be extracted to a file which can be run through any spell checking program. A separate utility is provided to merge the corrected text back into the source file. Since most spell checkers don't understand BASIC program statements and variable names, this feature could save programmers great embarrassment.

A Procedure Tree diagram may be generated showing the calling relationships between all SUBs and FUNCTIONs in your entire program. This feature will help you visualize the overall structure of a modular program.

Object summary reports may be created showing all distinct object names, and the number of times each is used. This type of report will help you find objects that are infrequently used and could possibly be eliminated.

You may generate a cross reference table that shows where all specified objects are used in your program. This report shows not only the reference line numbers, but also where assignments were made and what procedures they occurred in. This kind of report is invaluable since it allows you to see the scope of your variables, and avoid subtle bugs caused by variables that are referenced but were never assigned or vice versa.

Probably the most important report is one that lists all objects that are not used in your program. This will help you easily find mistyped variable names, unreferenced line labels, as well as procedures that are never used. One of the great features of BASIC is that you don't have to declare variables before you use them. While this is a great help when writing code, the compiler does not generate any warning when a variable is misspelled or unreferenced. This type of bug can lead to subtle problems that are extremely difficult to identify and locate. By providing an alphabetized listing of all unused variables, you can quickly spot any that are misspelled.

Another report lists all objects that are used within procedures, including their type and information about their scope. (A variable's scope means if it is private to a procedure, shared among several subprograms and functions, or common across an entire application.) This report can help you identify automatic variables (stack variables that are created when the procedure is called) and static variables. This will help you weigh the trade-off between speed and memory usage for a given procedures.

The last report type lists all variables that are used within a range of line numbers, but are also used outside that range. If you have ever wanted to take a section of code and move it into a sub program but where afraid that you would forget to also include related variables, this report will remove all of the guess work.

## HOW IT WORKS

The cross-reference program uses pulldown menus and dialog boxes to set the various options for its reports. After a BASIC source file and one or more report types have been selected, the program will read the entire source file as well as all associated files listed in the program's .MAK file. As the files are being read, the program parses individual objects, and saves them into several tables containing information about their type and usage. If specified, a formatted source listing will also be generated at this time.

After a source program has been read, all of the information about
each object will be saved to a file with the same base name as the
source file but with a different extension. The default extension is
.ODB (Object Data Base), however any extension may be specified if
you prefer. This file can later be used for other reports, without
requiring the entire source file to be read again. The cross-reference
program and this documentation refers to these files as *Object
Databases*.

Once the files have been read, the objects are sorted and the selected
reports are generated. If you specify that reports are sent to disk, you
may browse through the resulting files with the built in file viewing
feature.

## INSTALLATION

To install the cross reference program on your hard disk, create a
directory and log onto it. Then run the INSTALL program from the
distribution disk. For example, to install the files in a directory called
"\XREF" on drive "C:" type the following:

```
C:\> MD \XREF
C:\> CD \XREF
C:\XREF> A:INSTALL
```

Use the same method to install the source files, for example:

```
C:\> MD \XREF\SOURCE
C:\> CD \XREF\SOURCE
C:\XREF\SOURCE> A:SOURCE
```

If an error occurs during installation you should copy the distribution files directly to the drive and directory and then type:

    INSTALL

If you are installing or reinstalling an upgraded version of the software, you may want to include one of the following switches in the examples above:

    -n = Install newer files only
    -o = Overwrite existing files

If -o is not given, you will be prompted to overwrite each file that exists in your drive and or directory.

## REQUIRED FILES

After installation, the following files will be placed on your disk:

**File Name**

| | |
|---|---|
| README | If present, contains updated information about the program. |
| XREF.EXE | The executable version of the cross reference program. |
| XREF.KEY | Support file containing a table of BASIC keywords and other information used by XREF.EXE. |
| TEXTIN.COM | Text merging program. |
| TEXTOUT.COM | Text extracting program. |
| LINESOUT.COM | Line number removal program. |

**BASIC Source Files:**

| | |
|---|---|
| XREF.BAS | Source for the cross reference program's main module. |
| XREFDIAL.BAS | Source for the dialog box control module. |
| XREFMISC.BAS | Source for miscellaneous support routines. |
| XREFRPT.BAS | Source for the reporting module. |
| QMAP.BAS | Source for the reading and parsing module. |
| DIALOG.BAS | Source for the dialog box module. |
| GETFILE.BAS | Source for the file dialog box module. |
| DIALTYPE.BI | Type definition for dialog boxes. |
| XREFTYPE.BI | Type definitions for XREF. |
| XREFCOMM.BI | Defines common arrays used in XREF. |

| XREF. | MAKE or NMAKE description file for recompiling and linking the XREF program and its associated modules and libraries. |
| XREF.MAK | Lists required modules for XREF. |
| XREF.RSP | LINK response file for linking XREF. |
| XREFMISC.LIB | Library of miscellaneous routines used by XREF. |
| VIEWFILE.OBJ | File browsing module. |
| TEXTIN.BAS | Source for TEXTIN.COM, the text merging program. |
| TEXTOUT.BAS | Source for TEXTOUT.COM, the text extraction program. |
| LINESOUT.BAS | Source for LINESOUT.COM, the line number removal program. |
| MAKEKEYS.BAS | A program that creates the XREF.KEY file. |
| KEYWORDS.BI | Include file containing keyword information for MAKEKEYS.BAS. |

Note that XREF.EXE and XREF.KEY are the only files which are required to run the cross-reference program, and they must be kept in the same drive and directory.

If you intend to recompile XREF, you will also need the following files from Crescent Software's QuickPak Professional version 3.00 or later.

    PULLDNMS.BAS
    PRO.LIB

Notice that TEXTIN, TEXTOUT, and LINESOUT were linked using our P.D.Q. product. If you intend to recompile these utilities, the resulting .EXE files will be much larger when regular QuickBASIC is used.

## 2. PROGRAM OPERATION

This chapter contains information about the operation of the XREF.EXE program.

### OPERATION OVERVIEW

The usual sequence of events involved in using this program are as follows:

1. Start the program from DOS, either with or without command line arguments. (These arguments are described below.)

2. Select a program source file to examine, or a previously saved object database file.

3. Select the types of reports you want to generate.

4. Select the destination of the reports, either to a file or a printer.

5. Start the reporting process.

6. If you are reporting to a file, you may browse the file.

After you have made selections for the various reports and configured the program, you can save the current default settings to a disk file. This will eliminate having to re-configure the program each time you run it again later.

## COMMAND LINE ARGUMENTS

When you start the XREF program from DOS, the following options may be given:

    XREF filename[.bas], CFXFile[.cfx] [/b][;]

Filename is the name of the source file to process. The extension is optional and will default to ".BAS".

CFXFile is the name of a previously saved configuration file. The extension is optional and will default to ".CFX".

/b is an optional command switch which will force monochrome colors for all displays.

A semicolon (;) may be used to terminate a line, and it also causes the program to immediately start processing the source file (if given) with the current settings and defaults. If the semicolon is omitted, the menu will be displayed as usual. (Note: Any parameter may be omitted, but the CFXFile must always be specified after the comma.)

Please understand that command line arguments are strictly optional, since all parameters can be selected from within the program except /b.

## USING THE PULLDOWN MENUS

XREF's user interface is based on a comprehensive system of pulldown menus and dialog boxes. The menu system organizes the major settings categories as menu titles and pulldown subtitles. Dialog boxes query the user for additional information for certain categories.
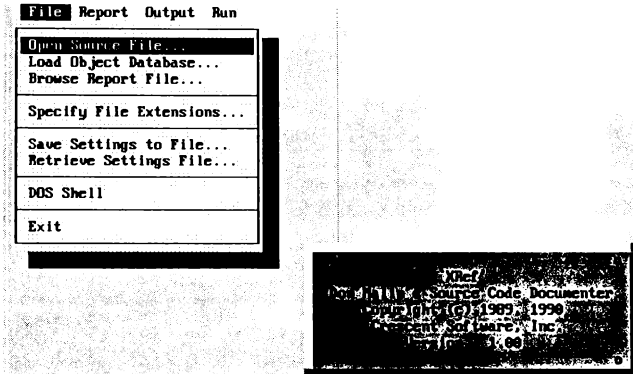
Displays source files to be opened and processed.

**Figure 1**

Figure 1 depicts XREF's introductory display with an active menu system. (The pulldown menu system is active whenever a menu is pulled down.) On the first line of the screen appears the menu bar, and under each menu bar option there is a unique pulldown menu. All of the options that are available in the program may be accessed through the menu system.

The table below summarizes the pulldown menu features. These
features are fully discussed beginning on page 20.

```
┌──────────────────────────────────────────────────────────┐
│                  PULLDOWN MENU FEATURES                  │
│                                                          │
│   Menu      Description                                  │
│                                                          │
│   File      Specifies files to be loaded for processing or │
│             viewing. Allows you to specify default extensions │
│             for files created by the program, allows saving of │
│             settings to a file, executes a DOS Shell, and exits │
│             the XREF program.                            │
│                                                          │
│   Report    Allows you to specify global report options as well │
│             as individual report types and objects to be listed │
│             in them.                                     │
│                                                          │
│   Output    Sets the destination for all reports as well as │
│             formatting information.                      │
│                                                          │
│   Run       Starts the reading and/or reporting process. │
│                                                          │
└──────────────────────────────────────────────────────────┘
```

The menu system reflects a user interface with which you are most
likely already familiar. Very much like QuickBASIC's own menu
system, XREF's menus may be used with either the keyboard or a
mouse.

The keyboard interface to the menu system is very extensive. In
general, the direction keys are used to select a menu and a pulldown
choice.

When XREF begins it presents the File pulldown menu depicted in
Figure 1. At this point you may scan across the menu bar by using
the Left and Right-arrow direction keys. Once the desired menu
(such as File) is selected, you may press the Up and Down-arrow keys
until the desired selection is highlighted.

Once a choice is highlighted, you may press Enter to select it. Some menu choices are indented, meaning they can be toggled on or off. To mark an indented choice as being either selected or unselected, press the space bar until a check mark appears or disappears. When a choice title is followed by three periods, it means that further information may be entered through an accompanying dialog box. Press Enter to call up the associated dialog box.

Each active menu choice will have a highlighted letter in its title. Inactive choices are those that are inappropriate in certain situations. For example, it would be inappropriate to start the reading and reporting process until a source file has been chosen. In this case the choice within the Run menu will be displayed in a low intensity color with no letter highlighted. Menu choices that do contain a highlighted letter can be selected by holding down the Alt key, and pressing the key corresponding to the highlighted letter.

If you have a mouse, you may use it to control the menu system. You can move the mouse cursor over a menu item and press the left mouse button. This will cause the desired menu to be pulled down displaying all the choices available in that menu. To make a selection from a menu, move the mouse cursor to the desired choice and press the left mouse button.

Some users prefer to "drag" the mouse. You may move the mouse cursor over the desired menu bar title and press and hold down the left mouse button. You may then select different pulldown menu commands simply by moving the mouse cursor along the pulldown menu. If the mouse button is released while the mouse cursor is over a choice, then that choice will be executed. You may also drag the mouse cursor along the menu bar to view other pulldown menus before making a selection.

If you do not wish to make a selection after you have activated a menu, simply move the mouse cursor away from the menu and release the left mouse button.

## USING THE DIALOG BOXES

A pulldown menu choice followed by ellipses (. . .) usually generates
a dialog box. Dialog boxes provide an effective way to gather
information from the user and make it easy to enter information or
select options.
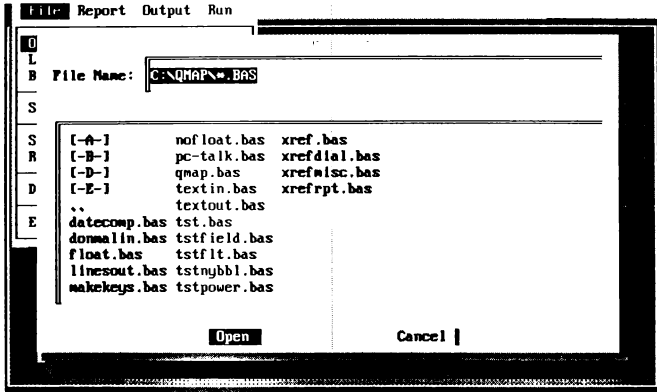


**Figure 2: The Open File dialog box**

Figure 2 depicts the Open File dialog box showing the three major
dialog box input elements: the text box, list box, and command
buttons. The text box accepts a string of characters from the
keyboard, and allows the entry of a path and file name. The list box
presents items in a columnar list, and shows all of the files that match
the file specification shown in the text box above. List boxes may
hold many items, and their contents can be scrolled by using the
cursor direction keys. The command buttons then carry out the
designated command when chosen. Pressing Alt-C using the example
in Figure 2 above will cancel the dialog box.

The table below presents a brief summary of XREF's dialog box input elements. These input elements, like the menu system, have both a keyboard and mouse interface.

When a dialog box is first presented, the cursor will rest on a particular input element. This cursor, or input focus, may be moved to the next input element by pressing the Tab key, or moved to the previous input element by pressing Shift-Tab. The input focus may also be directed to a particular input element by pressing Alt plus the first letter of a dialog input element label.

---

### SUMMARY OF DIALOG BOX INPUT ELEMENTS

| Input Element | Function |
| --- | --- |
| **Text box** | Collects text or numeric information from the keyboard. |
| **List box** | Presents a list of items usually arranged in columns. |
| **Check Box** | Allows an option to be turned on or off. When a check box is selected a check mark appears inside it; otherwise the check box is empty. |
| **Command Button** | Executes the command on the command button label. Once a command button is highlighted, pressing Enter will execute it. |

---

Aside from these general directions, there are more specific ways of using each dialog box input element with the keyboard:

   * Text box

The text box accepts text which is typed by the user, and it asks for specific information. When text is selected the entire string shown will be cleared when you begin typing. If you wish to edit the string without clearing it, you must use the left or right direction keys before typing.

   * List box

List box items are selected with the direction keys. When the desired item is selected you may press Enter to accept it.

   * Check box

The check box is toggled by pressing the space bar.

   * Command button

You may execute the highlighted command button at any time by pressing Enter. You may also use Tab to get to a particular command button and press the space bar or Enter. Further, pressing Alt plus the first character of a command button will also execute it.


If you have a mouse you may access dialog input elements by clicking on a desired element. More detailed instructions are summarized below.

   * Text

The mouse is not useful for entering information into a text box. You may, however, direct the input focus to a text box by clicking on it.

    \* List box

A list box item may be selected by *double clicking* on it. Double clicking refers to pressing the left mouse button twice in rapid succession. If a list box contains more information, its contents may be scrolled by clicking the mouse on a border.

    \* Check box

The check box is toggled by clicking on it with the mouse.

    \* Command button

You may execute a command button by clicking on it with the mouse.


## PROGRAM OBJECTS AND THEIR TYPES

The program and this documentation often uses the term *Object* to refer to distinct program elements. These elements are any of the following types:

| | |
|---|---|
| **BASIC Key Words** | Any BASIC key word or combination of keywords that forms a BASIC phrase such as INPUT or LINE INPUT. |
| **DEF FN Functions** | Any BASIC function name defined with the DEF FNname construct. |
| **FUNCTIONs** | Function procedure names. |
| **SUBprograms** | Subprogram procedure names. |
| **Labels** | Line numbers or alpha-numeric line labels. |

| | |
|---|---|
| CONSTants | Numeric constants such as the number 1, or named constants such as CONST True = -1 where True is a named constant. |
| Simple Variables | Any numeric or string variable that is not an array. |
| Simple TYPE Variables | An aggregate variable made up of elementary BASIC data types.  The XREF program only reports references to the type name, not the individual elements. |
| STATIC Arrays | Arrays that are allocated by the compiler. STATIC arrays are defined with a DIM statement, and use constants for the subscripts. |
| STATIC TYPE Arrays | STATIC arrays of a TYPE structure. |
| Dynamic Arrays | Arrays that are allocated at run time.  Dynamic arrays are defined with REDIM, or DIM with a variable for the subscripts. |
| Dynamic TYPE Arrays | Dynamic arrays of a TYPE structure. |

## SPECIFYING OPTIONS

Before XREF can begin, you must specify a file to operate on and at least one report or task to perform.  You will probably also want to customize the program to meet your needs and personal preferences. To accomplish this you will use the menus and dialog boxes to specify files, reports, and options, and then save the options in a configuration file.  The remainder of this chapter is devoted to the details of each program option.

## CHOOSING SOURCE FILES

You must choose a source file to process before any operations can be performed. Two types of files may be used, as follows. The first type is any ASCII BASIC source file. If this type of file is selected, it will be read and parsed into it's individual objects, and then saved as an Object Database. The second type of source file is the Object Database that was created earlier from the BASIC source file. This type of file can be used for subsequent reporting, without having to reread the entire BASIC source file.

## OPENING A PROGRAM SOURCE FILE

The first choice on the File menu is titled "Open Source File...". Use this choice to select a BASIC source file to be read and processed. After selecting this choice, a dialog box will appear showing a list of source files to choose from. You can either highlight the desired file name from the displayed menu, or type it manually.

When "Start" is selected from the "Run" menu, the program will first search for an associated .MAK file with the same base name as the specified source file. If a .MAK file exists, the module names specified therein will also be read and processed. If no .MAK file is found, the program will process only the specified file. .MAK files are generated by QuickBASIC, and they simply contain a list of program modules that are associated with the main module. $INCLUDE files are also read and processed as they are encountered in the source program.

Note that all modules must have been saved in "Text" format. QuickBASIC "Fast Load/Save" files cannot be processed. Also note that your programs must be syntactically correct. Programs that will not compile because of syntax or other errors cannot be processed properly.

## LOADING AN OBJECT DATABASE

The second choice on the "File" menu titled "Load Object Database..."
is used to display a dialog box containing a list of existing object
databases.  Use this choice to specify that you want to use a
previously created object database file instead of having the program
read your source file.

Object databases are generated each time a BASIC source file is read.
They have the same base name as the BASIC source's main module,
but with an .ODB extension.  These files contain a list of program
objects and information about their type and usage within the
program.  If you have not changed your program since the last time
XREF was run, you can generate reports based on the object database
instead of the actual source.  This will be considerably faster than
having XREF read through the entire program again.

## SPECIFYING FILE EXTENSIONS

The "Specify File Extensions..." choice from the "File" menu allows you
to change the default file extensions used for files created by XREF.
If the default extensions conflict with those of your own, you can use
this choice to change them.  After selecting this menu choice, a dialog
box will appear allowing you to fill in the following information.

The first field in the dialog box is titled "Object Database Extension:".
After a program has been read and analyzed, the information about all
the objects in the program will be saved to a database with the same
base name as the source file and the extension shown below.  This
makes it possible to run different reports without having to reread the
entire program.  If you don't want to save this information, enter
.NUL in this field.  As supplied, the default extension is .ODB.

The second field is titled "File Output Extension:". When you specify that report output should go to a file instead of the printer, the output file will have the same base name as your source file but with the extension shown. As supplied, the default extension for output files is .XRF. Note that you can override the default file name for any report by using the dialog box under the "to File..." choice on the "Output" menu.

The last field is titled "Extract Source Text File Extension:". While a source file is being read, the program can extract all quoted strings and/or comments, and place them into a file with the same base name as the source file but with the extension shown. This feature can be used to check the spelling in your programs, since most word processors do not understand BASIC programs and key words. As supplied the default extension is ".SPL".

## GLOBAL REPORT OPTIONS

You can specify certain options that apply to all reports, such as whether or not to display the source code as it is being read, or whether to use the IBM graphic characters in reports and line numbering conventions in reports. To specify these options, choose "Options" from the "Report" menu. After selecting this choice, a dialog box will be displayed allowing you to enter your preferences as described below.

## DISPLAYING SOURCE WHEN READING

The first input element titled "Display Source When Reading" is a check-mark field. If you want to see the source displayed on the screen as it is being read and analyzed, check off this field. Note that not displaying source will allow XREF to read and analyze the program faster.

## USING GRAPHICS CHARACTERS IN REPORTS

Selecting the input element titled "Use Graphics Characters in Reports" will cause all reports to include high-order (> 128 ASCII) line characters for dividing lines, headers and tree diagrams. Some printers use a different extended character set than the one IBM uses for video and printing. If the field is left blank, conventional ASCII characters will be substituted which will work on any printer.

## LINE NUMBERING CONVENTIONS

The field titled "Make Line Numbers Relative To:" is a list box containing three choices as follows:

"No Line Numbers" specifies that line numbers should not be printed on source listing reports. Reference line numbers will be given relative to the beginning of the file for other reports.

"Beginning of File" causes line numbers in all reports to be relative to the beginning of each source file (physical line numbers).

"Procedures" specifies that line numbers shall be relative to procedures. Each subprogram or function will begin on line 1. Note that this is the same method used within the QuickBASIC environment.

Most reports use line numbers to show where an object was used in a program. If you use the QuickBASIC environment for most of your editing, you should specify that line numbers be relative to procedures, since this is the way they are displayed on the bottom right corner of QuickBASIC screens. If you use any other editor you should probably specify one of the first two choices.

## SELECTING REPORTS

The "Reports" menu allows you to select various report types to be performed when the source program or object database is read. To select a report type for processing, move the highlight bar to the desired item and press the space bar. This will toggle a check mark on or off. Checked reports will be included in the program's output. Note that any or all options can be selected. To view or change options for a report, press the Enter key to view and edit the associated dialog box. The following sections describe the various reports and their options.

## SOURCE LISTINGS

Checking the menu choice titled "Source Listing..." causes all source code to be listed to the output device when the file is read. To specify this option on the pull down menu, press the space bar when this item is highlighted. If you press the Enter key, a dialog box will be displayed letting you specify the following information:

The first field in the dialog box can be used to enter a page title for each page of the listing. The program will automatically include the source file name, date, and time in all page headers, so you don't need to include this information here.

The second field is used to specify that all procedures will be listed beginning on separate pages. Press the space bar to toggle the check mark on or off.

The third field, "Expand tab characters to:", allows you to specify the number of character positions for each Tab stop. As Tab characters are encountered in the source file, they will be expanded to the next Tab stop. As supplied this is set to 8, but you should set this to the same value selected in QuickBASIC or whatever editor you use.

BASIC metacommands can be used within your source file to override XREF's defaults, or to control the format of the listing. The supported metacommands are listed below and each is described in detail in Appendix B.

<div style="border:1px solid">

### TABLE OF SUPPORTED METACOMMANDS

| Metacommand | Action |
| --- | --- |
| **$LINESIZE:n** | Tells XREF to change the maximum line width for the listing. |
| **$LIST** | Turns the listing of the source code on or off. |
| **$PAGE** | Forces a new page in the listing. |
| **$PAGEIF:n** | Skips to the next page if there are less than "n" printable lines left on the current page. |
| **$PAGESIZE:n** | Sets the number of physical lines per page in the source listing. |
| **$SKIP:n** | Skips "n" printable lines, or to the end of the page, whichever comes first. |
| **$SUBTITLE:'xxx'** | Sets a subtitle for listing page headings. |
| **$TITLE:'xxx'** | Sets a title for listing page headings. |

</div>

Although source listing metacommands have been supported in all Microsoft BASIC compilers since BASCOM 1.0, they are no longer documented. Appendix B shows both the use and syntax for each metacommand listed above.

## EXTRACTING SOURCE TEXT

Toggle the "Extract Source Text..." menu item on to specify that all quoted strings and/or comments should be written to a file when your source program is read. This feature makes it easy to check the spelling of all text used in a program.

You can specify saving either quoted strings, remarks, or both. Extracted text will be written to individual files with the same base name as the source file, but with an extension of .SPL (or whatever was specified under "Specify File Extensions..."). Note that .SPL files will also be created for any $INCLUDE file that is encountered. If an $INCLUDE file has the same base name as a module name, the .SPL files may conflict. Note also that .SPL files contain line numbers to allow edited files to be merged back in with the TEXTIN.COM utility.

See Appendix C which contains a description and instructions for the TEXTIN.COM and TEXTOUT.COM programs

## PROCEDURE TREE DIAGRAMS

Select the "Procedure Tree..." menu choice to specify that a Procedure Tree report be generated. This report type will show all procedures and their dependencies in a tree format that is easy to view.

Pressing Enter on this choice will display a dialog box allowing you to set the following options:

**Sort Procedures** - Procedures can be sorted such that they appear alphabetically, and only once on any branch of the tree. If they are not sorted, they will appear in the order and frequency in which they were used. Press the space bar to toggle this selection.

**Show procedure detail only once** - Procedures that call many other procedures and are in turn called many times can take up a great deal of room in a report. Check off this box to show the detail for each procedure only once in the report.

**Insert page breaks in report** - Checking this field will cause page breaks and headings to be inserted within the tree diagram.

Procedure Tree Diagrams are an ideal way to visualize the interrelationships of procedures within an entire program.

## OBJECT SUMMARY REPORTS

The "Object Summary Report..." choice tells the program to show all
specified object types, with information about where they are defined
and the number of times they are used.

If you press Enter on this choice, a list of object types will be
displayed from which you can choose the types of objects to be
reported on.

Object Summary reports can quickly show you all objects used within
a program, along with their frequency of use. This type of report is
very handy for finding variables that are infrequently used and could
possibly be replaced with a common scratch variable, thus reducing
the amount of near memory (DGROUP) used by your program.

If you specify that BASIC key words are to be listed, you will be able
to see all of the BASIC commands used within your program. BASIC
key words that require floating point math routines will be flagged,
allowing you to see if you are unnecessarily bringing in the floating
point portion of the BCOM library. QuickBASIC 4.5 and BASIC 7
PDS will exclude this code if your program doesn't use any floating
point keywords.

If you own Crescent Software's P.D.Q. package, you will also be able
to see if your program is using any keywords that are not supported.

## OBJECT CROSS REFERENCE REPORTS

The "Object Detail Report..." choice indicates that a detailed cross reference report should be generated. This type of report lists all specified object types, along with where they exist in the program and where they are referenced.

Each reference will be listed as a physical line number relative to either the beginning of the file or the beginning of the procedure it was used in, depending on what line numbering convention you specified in the "Options" dialog box. An equal sign (=) will precede line numbers where variables are assigned.

If you press Enter on this choice, a list of object types will be displayed from which you can choose the types of objects to be reported on.

This report is very useful for showing both the range and scope of variables in your program. This will help you determine where variables are assigned, and where they are referenced throughout the entire program. SHARED and COMMON variables will be shown, along with the names of the modules and procedures they were used in.

## REPORTING UNUSED OBJECTS

One of the most important uses for a cross-reference program is to find objects that were defined but never used. This type of error can be caused by misspelling a variable's name, or by editing a section of code that formerly referred to it.

Many languages require you to declare variables before you use them, however BASIC imposes no such restrictions. Although BASIC lets you write code more fluidly without interrupting your thought process, it doesn't check for misspelled or un-referenced objects. Once this type of bug has been introduced into a program, it can be very difficult to track down, and may cause unpredictable results.

To find un-referenced objects, check off the "Unused Objects..." choice from the "Report" menu. When chosen, this report will list all selected objects that are declared or created but never used. Procedures that are declared but never used, and constants (except numeric constants) or variables that are referenced only once will also be listed.

If you press Enter on this choice, a list of object types will be displayed from which you can choose the types of objects to be reported on.


## LISTING OBJECTS BY PROCEDURE

Objects may be listed by procedure so that you can see all specified objects that are used within a main module, subprogram, or function. Choose the menu item titled "Objects Used in Procedures..." from the "Report" menu to specify this type of report.

If you press Enter on this choice, a list of object types will be displayed, from which you can choose the types of objects to be reported on.

The report listing will show each procedure name followed by a list of all specified objects used within it. Variables will be listed along with an attribute word to the right of their names. The attributes are as follows.

## TABLE OF VARIABLE ATTRIBUTES

| Attribute | Description |
|---|---|
| **Static** | A local variable whose value will be retained between each invocation of the procedure it is contained in. Static variables are the default when a procedure is defined with the STATIC key word at the end of the procedure definition line. |
| **Automatic** | A local variable whose value will be initialized on each invocation of the procedure it is contained in. This is the default when a procedure does not have the STATIC key word at the end of the parameter list. This type of variable requires more stack space and more instructions to be executed each time the procedure is called. The advantage of automatic variables, however, is that the space they occupy when a procedure is active is reclaimed on exit thus saving precious memory. |
| **Parameter** | A variable listed in the procedure's parameter list that it is contained in. |
| **Shared** | A variable that is shared with the main module it is contained in. |
| **Global** | A variable that is shared throughout the module or program. |

This type of report can help you weigh the tradeoffs between static and dynamic (automatic) procedures, as well as see clearly what objects are being used in a procedure.

## LISTING OBJECTS EXTERNAL TO A RANGE

If you have ever needed to break up a large procedure into separate subprograms, functions, or modules, then the "List Objects External to a Range..." report will help you greatly. With this report you can specify a range of line numbers to examine, and the report will list all variables that are used within the range and also used outside it. Given this information you can safely move the section of code into a separate procedure or module, and use the report list in shared statements, or a parameter list to give your new procedure access to these variables.

To select this type of report press Enter on the menu choice titled "List Objects External to a Range...". A dialog box will be displayed where you can enter the following information:

The first two input elements are used to enter the beginning and ending line numbers to consider. Note that *line numbers* refers to physical line numbers, and not line numbers that are used as labels in your source code. To determine the line numbers while in the QuickBASIC editing environment, simply move the cursor to the top line of the range to consider, and note the first number shown on the bottom right corner of the screen. Then move the cursor to the last line to consider and note that line number as well.

If you specified that line numbers be relative to procedures in the "Options" dialog box as QuickBASIC does, you should enter the name of the procedure that contains the range in the third input element. If the range is contained in the main module instead of a procedure, or you specified that line number be relative to the beginning of the file, you should instead enter the name of the module that contains the range.

The remaining fields are check boxes where you can specify the types of objects to be reported. Also, be sure to see the Tips and Tricks section for more on this report type.

## OUTPUT OPTIONS

The Output menu allows you to select the destination of reports. Reports can be sent to either a file or printer. To select an output destination, move the highlight bar to the desired item and press the space bar. This will toggle a check mark on or off. Note that only one destination can be selected at a time. To change options for a printer, press the Enter key to view and edit the associated dialog box. The menu options are as follows:

## SENDING REPORTS TO A FILE

Selecting the first choice ("To File...") will send the report output to the file you specify, after pressing Enter on this choice. If you don't specify a file name, the reports will be written to a file with the same base name as your source file, but with the extension specified in the "Specify File Extensions" choice from the "File" menu. Any printer set-up codes or margin information from the "LPT1:" choice below will be sent to the output file.

## SENDING REPORTS TO A PRINTER

**"To Printer on LPT1..." or "LPT2..."**

This choice specifies that report output be sent to the printer connected to printer port 1 (LPT1:). If you press the Enter key on this choice, a dialog box will be displayed allowing you to set various printer options as follows:

## PRINTER SETUP

**Page Width:**

This specifies the character width of the printer. If you have a wide carriage printer or are using compressed printing, you should set this to the maximum number of characters per line. Otherwise, set this field to 80.

**Left Margin:**

Use this field to specify where printing should begin horizontally on the page. Entering 1 will cause printing to begin at the leftmost part of the page.

**Page Length:**

This option specifies the number of lines per page for your printer. Normally this should be set to 66, since standard letter paper uses this size. To suppress page breaks, enter 0. Note that this is the actual number of lines on the sheet of paper, and not the number of lines that will be printed on each page.

**Bottom Margin:**

The bottom margin specifies the number of lines to skip at the bottom of each page. The default is 6, but you can enter anything between 2 and the page length set above. The page length minus the bottom margin is the number of lines that will be printed before a form feed is issued. Note that footers will be printed two lines below the bottom margin.

**Printer Init. Code:**

This field is used to enter control characters that will be sent to the printer before the report begins. This makes it possible to set the printer into compressed or enhanced modes for reports. Low ASCII codes such as the Escape character may be entered by simply typing the ASCII number followed by a comma. Thus, you would enter "27, M" to send an Escape character followed by the letter M to the printer. If you had an Epson printer this would set enhanced (NLQ) printing mode. As supplied, this field will default to 15, which sets compressed printing on Epson compatible printers.

**Printer Reset Code:**

Enter your printer's reset code to restore the printer to its default condition after reports have been completed.

Settings for LPT1: and LPT2: are saved separately so you can maintain both printers independently.

# SAVING AND RETRIEVING CONFIGURATION FILES

### "Save Settings to File..."

After you have specified all the file and reporting options such as file extensions, report types, and printer information, choose this item to save the information to a configuration file. When the program is next run, this file will be read automatically with the same settings in effect.

You may also save and load multiple configuration files with different names. When "Save Settings to File..." is chosen from the "File" menu, you will be prompted for the name the file is to be saved as. If you use the default name XREF.CFX, the program will always load this file when it starts. If this file is not found, XREF will look for this file in its own directory. This makes it possible to have different configuration files for different directories, or one file only.

You can also save several configuration files under different names, which can be loaded using the "Retrieve Settings file..." choice from the "File" menu.

### "Retrieve Settings File..."

Use this choice to retrieve a previously saved configuration file. After selecting this, a dialog box will appear showing a list of configuration files to choose from. Simply highlight the one you want and press Enter.

## 4. RUNNING REPORTS

**"Run" menu, "Start"**

Once you have selected a source file (BASIC source or Object Database) and one or more reports, select this menu item to start the reporting process.

## BROWSING REPORT FILES

**"Browse Report File..."**

Use this option to view (browse) a previously created report file. When this menu item is chosen, a dialog box will be displayed allowing you to select a file to view. This feature is intended for viewing report files generated by the program, though it will also work with any ASCII text file. After selecting a file from the file name dialog box, the contents of the file will be displayed. Use the cursor direction keys to move through the file.

## INVOKING A DOS SHELL

**"DOS Shell"**

This choice temporarily exits the program and invokes a DOS shell. Use this choice when you want to access DOS without leaving the program. When you are finished in DOS, type EXIT to return to XREF.

## LEAVING THE PROGRAM

**"Exit"**

This choice exits the cross reference program and returns to DOS.

## 5. PROGRAM LIMITS

XREF can keep track of a maximum of 2,425 distinct objects and 32,766 total references. Depending on the amount of memory available to the program, these maximums may be less. If your source files have more of either maximum, the program will abandon the reports and display an error message. In this case you could remove one or more modules from the main program's .MAK file and run the report again.

## 6. TIPS AND TRICKS

Aside from the obvious uses for the various reports included with the package, there are some other interesting uses we have found while using XREF. The following are some useful tips and techniques that you can use to improve your programs.

### EXCLUDING THE FLOATING POINT LIBRARY

With the introduction of Microsoft QuickBASIC 4.5 it is now possible to create programs that do not include the floating point math library if your program does not need it. This can result in a significant size reduction of between 3K and 10K. While this is a great improvement over previous versions of BASIC, many programmers don't know that they are unnecessarily using keywords that automatically include this library.

To check your program for keywords that bring in the floating point library, use the "Object Summary Report" to list all BASIC keywords as follows:

After starting the XREF.EXE program, select "Open Source File" from the "File" menu, and then enter or highlight the name of your main module. Next, go to the "Report" menu and highlight the choice titled "Object Summary Table", and press Enter to view the associated dialog box. Check the first field titled "BASIC Key Words", and optionally un-check all the rest of the fields. Finally, press Enter to accept the settings. After selecting the report, choose a report destination from the Output menu, and start the reporting process.

Once XREF has completed its work, you can examine the generated table. Each BASIC key word used in your program will be listed, along with the number of times it was used to its immediate right. Each key word that requires the floating point library will have an "f" character immediately to its left. Some keywords may also have a "p", indicating that it is not supported by Crescent Software's P.D.Q. library. Once you have identified all floating point keywords, you will need to find alternative methods or commands to replace them.

A commonly used key word (operator) that requires floating point support is the caret (^), which raises a numer to a power. Many programmers use this function to test or set bits within an integer or long integer variable, thus allowing one variable to represent many individual flags. The following is an example of this technique:

```
Bit% = 3                              'Use the 3rd bit
Number& = Number& OR 2 ^ Bit%         'Set the bit
Number& = Number& AND (NOT 2 ^ Bit%) 'Clear the bit
IF Number& AND 2 ^ Bit%) THEN         'Test the bit
   'it's set
END IF
```

While you might think that this would not require floating support since only integers are used, BASIC will actually convert the integers to floating point values, and call a floating point routine to do the math. The Power operation in these statements can and should be replaced with a function such as the one shown below.

```
    Bit% = 3                                    'Use the 3rd bit
    Number& = Number& OR Power2&(Bit%)          'Set the bit
    Number& = Number& AND (NOT Power2&(Bit%))   'Clear the bit
    IF (Number& AND Power2&(Bit%)) THEN         'Test the bit
        'it's set
    END IF
    .
    .

FUNCTION Power2& (Power%) STATIC
    TEMP& = 2
    FOR N = 2 TO Power%
        Temp& = Temp& * 2
    NEXT
    Power2& = Temp&
END FUNCTION
```

The above example and function requires no floating point support, and will also be considerably faster.

Another example of an unnecessary floating point operation is using the forward slash division operator (/) with integer variables. If your program uses this divide operator you should change it to the integer divide operator (\) instead if at all possible.

Other BASIC statements that add the floating point library routines to your programs are PLAY, SOUND, LINE, and VAL. Besides the floating point routines, these statements are also relatively large, because they must be able to evaluate the entire range of floating point values. We provide a number of replacements for these in our QuickPak Professional product, solely to help you reduce the size of your programs.

## MOVING SECTIONS OF CODE INTO SUBPROGRAMS OR FUNCTIONS

During the development of a program you may decide that a section of code could or should be made into a separate subprogram or function. For example, any collection of statements which is called more than once and that exceeds, say, twenty or more lines of code is an ideal candidate for conversion to a subprogram or function. This lets you avoid duplicating similar sections of code. Another reason for creating subprograms is to hide the lower level details from the flow of the main program. This will improve the readability of your code, by letting you view the major tasks being performed, rather than being burdened with many individual details. Finally, if your programs are very large and the BC compiler itself is running out of memory, dividing your program into separate files will enable you to compile them individually, and link them together to create a final .EXE program.

Notice that very small subroutines are often better implemented as GOSUB routines, because a certain amount of overhead is required for each separate subprogram or function. Also notice that BASIC PDS version 7 will place subprograms and functions into expanded memory (EMS), but only if they can be stored in 16K or less each. Therefore, if you are using BASIC 7 and find that you are frequently running out of memory in the QBX envirnment, you should avoid creating many very large subprograms.

One of the problems in moving a block of code into a separate subroutine is that the code may refer to global variables elsewhere in your program. And that's where XREF can help. By using the "List Objects External to a Range" feature, XREF will list all of the variables that are not contained within a specified range of lines. The discussion below shows how to use this feature to successfully move a block of code into a subprogram or function.

First identify the section you want to move to the new procedure. From within the QuickBASIC environment or your own editor, record the line number of the first line of the section. Next record the number of the last line in that section. (Be sure to remember the name of the procedure that the program fragment is in. If the section is contained in the main part of your program, the procedure name will be MAIN.) Start XREF and select the name of your program from the "Open Source File" menu choice. Next, move to the Report menu and press Enter on the "List Objects External to a Range" choice. Within the displayed dialog box, enter the line numbers you recorded earlier, along with the procedure name in the appropriate fields. Also select the types of objects to report. For the example we are working on, you should choose all object types except constants, since constants are shared automatically. It is also a good idea to leave "Labels" checked, since you may be using a GOTO or GOSUB to a section of code outside the range you are moving.

After completing the dialog box, press the space bar on the menu choice to mark this report type as selected. Then choose a destination for the report from the Output menu. You may want to send the report to a file so that you can load it into QuickBASIC as a document, and cut and paste the list of objects between your program and the report. When all the options have been set, Choose "Start" from the Run menu to generate the report.

The created report will list all of the variables that need to be shared with your new procedure, or passed as parameters.

## MOVING PROCEDURES BETWEEN MODULES

It is often necessary to move procedures between modules when a module becomes too large to compile, or when you decide that it might be more appropriate to move it to another module containing similar or related procedures. In the QuickBASIC environment this can be done easily with the "Move" command in the "View - Subs" dialog box. The tricky part comes when your procedure shares data within the module it is being moved from. If the data is shared using a SHARED statement in the procedure, it is easy to see what needs to be done. However, if the data is global to the entire module, it is difficult to determine what variables are required by the procedure. Another problem arises when the procedure uses other procedures declared in its current module, but not in the destination module.

The solution to the first problem is to either use COMMON in both modules to share the data, or include the shared variables in the procedure's parameter list. Using COMMON is the easiest method, but it leads to interdependence between the modules. If the two modules are always used together and only by each other, this is fine. However, if you intend to use either module with other programs, you should try to keep it independent by avoiding common data.

In order to find out what variables or DECLARE statements need to be passed or used in the new module, you should use the report titled "Table of Objects Used in Procedures". This report will list all objects (both variables and other procedures) used by all procedures in your program, including both passed and shared variables as well as internal variables.

To generate this report, select the name of your main module from the "Open Source File" choice from the File menu, and then press Enter on the "Table of Objects Used in Procedures" choice of the Report menu. A dialog box will appear letting you indicate which object types are of interest to you. For the example above you should probably check all of the object types except BASIC keywords. After accepting the dialog box options, press the space bar to select this report type for processing.

When all options are set, choose "Start" from the Run menu to generate the report. The created report will list all variables used within each procedure, along with an attribute for each such as the word "SHARED" or "Global".

## CONSERVING DATA SPACE BY REUSING VARIABLES

Many programmers code a FOR/NEXT or WHILE/WEND loop using a variable name that is meaningful to the loop. For example, to read in a text file we've seen code like this:

```
FOR TextLine = 1 TO NumLines
    LINE INPUT #1, Work$
    PRINT Work$
NEXT
```

However, for each different integer variable that is used, two bytes are taken from available string memory. A much better approach is to use a temporary variable repeatedly, such as X or maybe I. If the same variable is always used, that much memory may be saved. This idea could even be extended to include subprograms and functions, by adding the following statement near the beginning of your main module:

```
DIM SHARED X, Y, Temp
```

Then, these variables can be used any time a FOR/NEXT counter is needed, or whenever a temporary scratch variable is needed.

Another memory waster is static variables used in subprograms and functions. When a variable has not been declared as SHARED (or COMMON for that matter) and it is used in a subprogram, it is distinct from other variables with the same name in the main program or in other subprograms. While local variables is definitely a major feature of modern structured languages, it is important to understand that each variable uses more memory.

Therefore, we often recommend omitting the STATIC option in SUB and FUNCTION definitions when possible. When STATIC is not used, all of the variables within a given subprogram are stored on the PC's stack temporarily, and only while the subprogram is active. Of course, this has the side effect of losing whatever value a variable held between invocations. However, you may specify STATIC for individual variables if necessary.

## ELIMINATING LINE NUMBERS

When your programs begin to approach the size and memory limits that the BC compiler can handle, it is good practice to remove any unnecessary line labels and line numbers. For each line that is labeled, BC must remember its position in the source file, to resolve GOTO and GOSUB statements that may reference it later in the program. Further, if you compile with the /d (debug) option, then all of the line numbers are added to the final .EXE program. XREF therefore includes the LINESOUT utility to remove all unused line numbers and labels.

To remove unused line numbers and labels from a program, first use the XREF program to create an Object Database for that program. The generated Object Database will contain information about each line number and label in your program, including the number of references to each. This is needed by the LINESOUT utility. Once an Object Database exists for your program, you can use LINESOUT.COM to actually remove the numbers and labels from your source. To use LINESOUT, type the following from DOS:

```
LINESOUT [filename[.odb]]
```

Where "filename" is the name of the Object Database file created by XREF. The extension is optional and will default to ".ODB" if omitted. If you do not specify a file name, the program will prompt you for one.

Once started, the program will read the Object Database and search for all unreferenced line numbers or labels while simultaneously reading the original BASIC source file. As lines of source are read, they will be written out to a temporary file with the extraneous labels removed. Once the entire source file has been read, the original source file will be renamed with a .BAK extension, and the new source will be written using the original file name. If a .BAK file already exists, you will be asked if you want to overwrite it. If your program uses more than one module, the next module will then be read until all of them have been processed.

Notice that LINESOUT was designed as a separate utility, rather than including its function in the XREF program. This was done because removing line numbers is usually done only once, and the extra code would unnessarily burden the XREF program if it was included. Also notice that LINESOUT was written using our P.D.Q. library to achieve the very small file size.

# 7. ABOUT THE PROGRAM'S SOURCE CODE

### RECOMPILING THE PROGRAM:

There is little reason for you to recompile XREF, since there are very few things that could or should be customized. However, in the interest of completeness, the following discussion explains each of the various program modules, and how they have been compiled.

XREF.EXE and all of its associated modules were compiled with the /o (stand alone) and /ah (array huge) switches to allow maximum memory usage for the large tables that are required. In addition, the source was compiled and linked with BASIC 7 to allow the use of overlays; again to save memory for the program.

The XREF program's source code consists of several overlaid modules including the main module, several user interface modules, a parsing/analyzing module, and a reporting module. The various modules and their basic functions are listed below.

## XREF.BAS

This is the main module, and it controls all aspects of the program's operation. The menu is set up and controlled from this module, as well as the control of the various dialog boxes, source reading, and reporting routines. Since the other modules are overlayed, some commonly used subroutines are also kept in this module, such as the printing and file name parsing routines.

Keeping them in the main module eliminates the problem of having to constantly swap them in and out of memory (or disk) as they are used.

## PULLDNMS.BAS

This module handles the actual operation of the pull-down menu system. Once the menus are set up by the main module, this module handles displaying the menus, and all menu movement within them. Note that the source code for this module is not included with XREF, but is part of our QuickPak Professional tool box.

## DIALOG.BAS

This module handles the operation of all the dialog boxes used by the program. Once a dialog box has been set up by the main (or other calling) program, the display and action of all dialog boxes are handled by this module.

## GETFILE.BAS

GETFILE sets up and manages the dialog box used to select files names within the program. It also handles the error prompt dialog boxes.

### XREFDIAL.BAS

This module sets up all the dialog boxes used in the program, except for the file selection and error prompt dialog boxes mentioned above.

### XREFMISC.BAS

This module handles loading and saving object databases.

### QMAP.BAS

The QMAP module is responsible for reading, parsing, and analyzing your source files. QMAP creates the tables of objects and their reference information that is used by all the reports. In addition, the source listing report is generated by this module.

### XREFRPT.BAS

This module contains all the various reporting routines used by the program.

The following $INCLUDE files are also required for some of the modules:

### DIALTYPE.BI

This file contains a TYPE definition for use with the various dialog boxes.

**XREFTYPE.BI**

This file contains TYPE definitions for program control data, as well as for the tables of objects and references.

**XREFCOMN.BI**

XREFCOMN defines all of the COMMON arrays and data that are used throughout the XREF program.

In addition to the above BASIC source modules, you will also need two libraries containing assembler routines used by the program. The XREFMISC.LIB library contains routines specific to the XREF program, and PRO.LIB contains various general purpose routines  PRO.LIB is not provided with this package, however it is included with our QuickPak Professional library.

We have also provided response files for use with Microsoft's MAKE utility, to automate the compiling and linking process. XREF. is the MAKE description file for building the program, and XREF.RSP is the LINK response file used by MAKE when creating the final XREF.EXE program. You can examine these files to see what compiler switches are used for each module, as well as the various "NO", or stub files that are used.

The XREF program was built with Microsoft BASIC 7 PDS compiler and linker in order to take advantage of its overlay capabilities, to maximize the amount of memory available to the program. If you do not have BASIC 7, you may use QuickBASIC versions 4.00b through BASIC 6. However, the resulting program will have less memory available to it, and thus a reduced object capacity.

# APPENDIX A

## PROBLEMS

**Problem:**

Incorrectly reporting file operations such as "CLOSE#1", "GET#1" etc. as variables, constants or line labels.

**Solution:**

There must be a space between the BASIC keyword and the pound sign.

**Example:**

"CLOSE#1" should be "CLOSE #1"

**Discussion:**

While the original syntax is valid for the BC compiler, we have chosen not to burden XREF with the extra logic required to handle this seldom used construct. If you use the QB.EXE or QBX.EXE editing environment for program development, this will never be a problem.

**Problem:**

Reporting variables listed in named common blocks as different variables.

**Solution:**

You are using different variable names in corresponding named common statements. Aliasing of variables between modules through named common blocks is not supported by the XREF program. You must use the same variable names across modules.

**ERROR NUMBERS**

It is possible that a fatal error may occur during the operation of the XREF program. If one occurs you will see a message of the form:

Error ## occurred in module...

The following is a list of error numbers and their probable causes:

| Error # | Possible cause |
|---------|----------------|
| 24 | Your printer is not turned on or is off-line. |
| 27 | Your printer is out of paper. |
| 61 | The disk that output has been directed to is full. |
| 67 | The program attempted to open more files than are available under the current "FILES=" statement in your CONFIG.SYS file. This usually occurs when your program uses nested Include files. |
| 68 | You specified a printer that is not available on your system. |
| 70 | The program attempted to write to a file that has been locked by another program or process. (This could occur on a network.) |

If you receive any error number other that those listed above, please note the circumstances under which the error occurred and call us.

Notice that while fatal errors may occur and stop the program, no damage will be done to any of your source files. The XREF program never alters source files.

## ERROR MESSAGES

The following is a list of program error messages that may occur during the operation of the XREF program:

`Can't find XREF.KEY`

XREF is unable to find the XREF.KEY file. This file contains information about all of BASIC's keywords, and it is necessary for the operation of the program. XREF.KEY must reside in the same drive and directory as the program file XREF.EXE. If you have DOS 3.0 or later, the program will be able to find this file no matter what directory you started the program from. If you have an earlier version of DOS, you will have to start the program from the same drive and directory in which you installed the program.

`Can't find configuration [filename]!`

The program could not find the specified configuration file. Either you mistyped it at the command line, or the file is not in the current drive and directory.

`Cannot process QuickBASIC Fast Load/Save files . . .`

The program tried to open a BASIC source file that has been saved in the QuickBASIC "Fast Load/Save" format. The XREF program cannot process this type of file. Note the name of the module, and then save the source file in ASCII text format from QuickBASIC and try again.

**File not found**

The specified file could not be located. If you typed the file name manually, you may have misspelled it, or perhaps the path was incorrect.

**[filename] is not a compatible configuration file**

You tried to load a configuration file that is not compatible with XREF. Either the specified file is not a valid configuration file, or it was created by a previous version of XREF. If the configuration file name is XREF.CFX, then the program tried to load an incompatible version on startup.

**[filename] is not an Object Database!**

You tried to load a file that is not an XREF compatible Object Database. You may have mistyped the filename, or tried to load a file that has been corrupted. Either retype the file name, highlight the filename rather than type it, or open a BASIC source file instead.

**[filename] not found!" or "Can't find [filename]!**

If XREF was using a .MAK file to locate associated modules, the designated file may not exist or it may be in another directory or drive. You should edit the .MAK file to remove the nonexistent file, or change its path name to include the correct drive and directory.

`[procedure name | filename] is not used by File Name!`

The procedure name or module name specified in the "External to Range" dialog box does not exist, or it is not used by your program. You may have mistyped the name when you filled in the dialog box.

`$INCLUDE file [filename] not found!`

The program cannot find a file that was specified in your source file with an $INCLUDE metacommand. Edit the $INCLUDE statement to reflect the correct drive and directory, or update your DOS environment to include the directory where you keep $INCLUDE files.

`Insufficient EMS to load overlays`

If your system has expanded memory (EMS) but less than 64k is available, the program cannot operate. You will need to either allocate more expanded memory, or disable it entirely. This situation can occur when you are using a 386 operating system such as QuarterDeck's DesqView, but the virtual machine you are running has no expanded memory allocated to it. In this case the program will see the expanded memory, but will not be able to use it for overlays. Try allocating more memory, or run the program in another virtual machine.

`Not enough memory`

There was not enough free memory to allocate object information tables. You need to free up more conventional memory for the program. You may be able to do this by removing TSR programs and or device drivers from memory.

**Too many objects**

The program encountered more objects than it has memory for. If the total number of objects displayed on the bottom line of the screen is less than 2,425, you must make more memory available to XREF and try again. You may be able to do this by removing TSRs and/or device drivers from memory. If the maximum number of objects (2,425) has been reached, you must remove one or more module names from the source program's .MAK file and try again. You may then run XREF against the removed modules individually if needed.

**Too many references**

The program encountered more references to objects than it has memory to accommodate. If the total number of references displayed on the bottom line of the screen is less than 32,766, you should make more memory available to the program and try again. You may be able to do this by removing TSRs and/or device drivers from memory. If the maximum number of objects (32,766) has been reached, you will have to remove one or more module names from the source program's .MAK file and try again.

## APPENDIX B

### IMBEDDED METACOMMANDS

Note that any changes to XREF's defaults made by imbedded metacommands will effect not only the source listing but also all of the remaining reports.

### '$LINESIZE

| | |
|---|---|
| Purpose: | Tells XREF to change the maximum line width for the listing. |
| Syntax: | '$LINESIZE: Number |
| Where: | Number is the last column to print at before wrapping takes place. |
| Comments: | The '$LINESIZE metacommand must appear within the first two lines of your program if you want the entire listing to be the same width. If '$LINESIZE appears anywhere else in the program, it affects only the width of the remaining lines. |

### '$LIST

| | |
|---|---|
| Purpose: | Turns the listing of the source code on or off. |
| Syntax: | '$LIST+ or '$LIST- |
| Where: | "+" turns the listing on, and "-" turns it off. |
| Comments: | '$LIST is useful for getting partial listings of new or modified code. Note that '$LIST+ will not turn a listing on unless "Source Listing" was selected from the "Report" menu. |

**'$PAGE**

| | |
|---|---|
| Purpose: | Forces a new page in the listing. |
| Syntax: | '$PAGE |
| Comments: | The page is forced by inserting a form feed character (CHR$(12)) into the listing, followed by a heading for the new page. |

**'$PAGEIF**

| | |
|---|---|
| Purpose: | Skips to the next page if there are less than "n" printable lines left on the current page. |
| Syntax: | '$PAGEIF: n |
| Where: | "n" is the number of lines to skip for each page, and it must be in the range of 1 to the page size. XREF uses 6 lines as its default. |

## '$PAGESIZE

| | |
|---|---|
| Purpose: | Sets the number of physical lines per page in the source listing. |
| Syntax: | '$PAGESIZE: n |
| Where: | "n" is the number of lines that occupy a page in the listing. XREF's default is 66. Pages in a listing are separated by a form feed character (CHR$(12)) followed by a page heading. |
| Comments: | If "n" is 0, no form feeds or subsequent headings will be printed. The '$PAGESIZE metacommand must appear within the first page of your program if you want all the pages to be the same length. If it appears anywhere else, it will affect only the remaining pages. |

## '$SKIP

| | |
|---|---|
| Purpose: | Skips "n" printable lines, or to the end of the page, whichever comes first. |
| Syntax: | '$SKIP: n |
| Where: | "n" is the number of lines to skip. Note that the last six lines, or the number set with '$PAGEIF are always skipped. |

### '$SUBTITLE

|            |                                                            |
|------------|------------------------------------------------------------|
| Purpose:   | Sets a subtitle for listing page headings.                 |
| Syntax:    | '$SUBTITLE: 'Text'                                          |
| Where:     | Text is the subtitle to be inserted.                       |
| Comments:  | The specified text is printed to the right of the file name in all subsequent page headings. |

The '$SUBTITLE metacommand must appear within the first two lines of your program if you want all of the pages to have the same subtitle. If it appears anywhere else, it will affect only the remaining pages.

### '$TITLE

|            |                                                            |
|------------|------------------------------------------------------------|
| Purpose:   | Sets a title for listing page headings.                    |
| Syntax:    | '$TITLE: 'Text'                                             |
| Where:     | Text is the title to be inserted.                          |
| Comments:  | The specified text is printed on the left side of the first line of all subsequent page headings. |

The '$TITLE metacommand must appear within the first two lines of your program if you want all of the pages to have the same subtitle.  If it appears anywhere else, it will affect only the remaining pages.

## APPENDIX C

SOURCE TEXT EXTRACTING AND MERGING UTILITIES

**Instructions for TEXTOUT.COM and TEXTIN.COM**

The two utilities TEXTOUT and TEXTIN can be used together to export, and then import, strings from and to a BASIC source file. This makes it possible to check the spelling of all quoted text and remarks in a program, or even to simplify translating to another language. The TEXTOUT program may also be used to create a file of program text that could be read by the program at run time, instead of keeping quoted strings in the source. This would save string space, since the strings would exist only in their variables, and not also in the quoted constants. (The statement [INPUT #1, X$] reads the string from disk and stores it in the string; [READ X$] makes a copy of the DATA string when it assigns it to X$. Therefore, the string is stored in memory twice.) To extract quoted strings, start TEXTOUT from the command line with the name of the source file as an argument as follows:

    TEXTOUT filename[.bas] [/c]

The program will then read and process the specified file. The /c option specifies that remarks as well as quoted strings are to be extracted. Notice that the source file extension is optional, and when omitted will default to ".BAS". As the program encounters each quoted string, they will be written to a file with the same base name as the source file but with an extension of ".SPL". Also note that zero length strings ("") will not be written, nor will strings that contain only spaces. When the program has completed reading the source file, it will return to DOS. At this point you may edit the .SPL file, or run it through any spell checking program.

CAUTION: If you intend to merge the .SPL file back into the source file, do not make any changes to the source file that would effect the sequence of lines, or alter the number or location of quoted strings. After the .SPL file has been edited, its text can be merged back into the source file using the TEXTIN utility. To do this, start the TEXTIN program from the command line with the name of the BASIC source file as an argument as follows:

```
TEXTIN filename[.bas]
```

The program will then read both the source file and the .SPL file, and replace all quoted strings (and remarks if /c was used when running TEXTOUT) from the source file with the strings from the .SPL file. If a remark follows a quoted string, its position will be retained if possible, regardless of the difference in lengths of the two strings. The modified source will be written to a file with the same base name as the original source file, but with an extension of ".TRN". After all the source has been read and written, the program returns to DOS.