¨WhizWare -- company that developed this software

¨WhizTool -- this software package and its on-line user's manual

¨Overall design philosophy -- why the tools look like they do

¨How they work -- general design strategy (an overview)

¨When to use -- which ones to use for various purposes

¨Limitations -- some restrictions (deliberate design tradeoffs)

¨LHITS -- Line Hits -- line numbers of lines "branched to" thus far

¨LXREF -- Line Cross Reference -- "go-to-come-from listing"

¨VFIND -- Variable Find -- reports lines where a name occurs

¨VLIST -- Variables List -- ones used thus far and in which order

¨VXREF -- Variables Cross Reference -- all named and their line numbers

|WhizWare

```
+------------------+
¦ WhizWare         ¦
¦ 635 Kendrick Rd  +--- A computer software development shop.
¦ Milner, GA 30257 ¦
+------------------+
```

The analyst that designed the ¨WhizTool programs is Thomas C. McIntire. Write to Tom if you have any problems or wish to comment on this software. He will appreciate the input, and will fire off a quick response to your questions. His E-mail address is: whizware@bellsouth.net

|WhizTool

WhizTool is our "catch all name" for this collection of five programs and the accompanying on-line user's manual, i.e., this program.

WhizTool.COM is the program that displays the manual, WhizTool.DAT.

Mouse lovers: This program checks to see if MOUSE.COM or MOUSE.SYS has already been loaded. If not, or Mickey is napping, this program simply reverts to no-Mouse mode. (Use the cursor arrow keys instead.)

> Also: Only one of two buttons are tested for; the left one works like <Enter> and the right one like <Esc>.

The five tool programs all have a file extension of dot-BAS; they are all stored in ASCII format (they were saved using the comma-A option on SAVE). Thus: The tools themselves are "mergeable modules"--they are made use of by merging them into end product programs during their development. Then, typically, they are deleted. Because they are all numbered alike they can also overlay each other. Read on ....

|Overall

The overall design: These programs are meant to be added on to the tail-end of a program already loaded into memory. The tools that analyze program lines then "read" from the first line of the real program, stopping when line 9000 is reached so that the tool itself is not included in the output.

To know when to stop depends on two things: The interpreter has a work area word that it keeps updating with the line number of the currently executing line. When we RUN or GOTO the first line of a tool, logic on that line saves its own line number. (So that if the program or the tool is renumbered, the tool can take care of itself.)

RUN is used to start the static code analyzers-- ¨LXREF ¨VFIND ¨VXREF -- because RUN lets the tool program itself build a new stack of variables of its own. (So the tool and the real program make use of the same memory.) GOTO is used to start the dynamic tools-- ¨LHITS ¨VLIST --so that the interpreter's work areas are still intact at the time the tool is started.

The internal logic of each program depends on a specific awareness of how GWBASIC.EXE works. That knowledge is thoroughly documented in ¨GeeWhiz -- the on-line language reference manual also available from ¨WhizWare

(continued)

|

Parsing along a line: A 255-byte string is used to hold codes--pointers for an ON GOTO, to a piece of logic that can handle a particular type of byte that might be encountered. This driver-string is constructed in a file buffer so that it is external to the interpreter's dynamic management of variables. A series of POKE statements is done during start-up to insert GOTO codes into these bytes appropriate to the needs of each tool.

Parsing of variables: The GOTO codes are set up to "step over" numeric words, key word tokens, quoted strings and most other can't-be-a-name bytes. Any byte in the decimal range of 65-90 (ASCII caps A-Z) is thus the first byte of a name. Maybe. A test string is built of this first letter, plus those that follow and any character from the set "0123456789.!#$%(". This suspect must then be compared to the "untokenized" key words list to decide whether it should be reported or not.

> Note: Nearly all key words in the language are stored as tokens (codes). Nine words are not, however. They are stored as ASCII caps-strings. They are ACCESS, ALL, APPEND, AS, BASE, OUTPUT, RANDOM, SEG, and SHARED. Also, the letter-switches B, and BF are peculiar. When used with graphics LINE statements they are differentiated from variables based on how they are used "in context". See ¨Limitations for more about these turkeys.

|How

All five of these tools are unique in what they can be useful for, but they are highly similar in how to make use of them.

They all begin at line 9000 and were saved with the A-for-ASCII option on SAVE so they can be merged into a program already in memory. And they can overlay each other. They can be gotten rid of by a simple DELETE 9000-.

Two or more of these tools can be in memory at once by merging-in one, then renumbering it to begin at, say, 7000. (Assuming enough memory, of course.) They can be left in during development, and be used repeatedly at various times, as well.

Output is normally to the screen, but PRINT can be easily changed to LPRINT statements if desired. Obviously they can be further customized, also, but specific knowledge of how GWBASIC.EXE works is needed. Again, that detail is thoroughly documented in ¨GeeWhiz --the on-line language reference manual also available from ¨WhizWare

|When

Each of these tool programs were devised for solving certain types of programming "problems". Some typical cases are listed below.

Need to coin a new variable? Run VFIND to check to see if a prospective name has already been used.

Want to change the name of an existing variable? Run VFIND to get a list of all of the line numbers that name exists in.

If a variable's contents is other than what you expected at a point, use VXREF. Check the lines that contain that variable. And check those for other variables that have similarly spelled names; maybe a typo is all that is wrong.

Logic-flow problems can nearly always be solved by using the output from LXREF, tediously tracking all branching activity, one line at a time. This is a "static" exercise however. Sometimes a dynamic approach will take less time. Run a program to a point--maybe until a temporarily inserted STOP statement. Check the output from LHITS to see if a particular branch has occurred, i.e., the target line number of a GOTO, GOSUB, THEN, etc. LHITS is especially handy for debugging event traps like ON TIMER, or on COM(n).

(continued)

|

The most common use of VLIST is for performance optimization. Every time a variable is encountered during execution the interpreter must run through its index of names already assigned. It does it from the top down, one name at a time, every time. We can shorten this search "overhead" by stacking names in a deliberate order, forcing the most frequently used names to be near the top of the index. (Names are added to the index when an area of memory must be allocated to hold their contents; each new name gets added to the bottom of the list.) VLIST lists the variable names index at any given point in time. This is an easier solution to the problem than trying to anticipate exactly which types of statements cause names to be added to the index. (Some manuals say that names are added by any first-using statement; that is an erroneous, over-simplification. The exceptions list is considerable.)

Quality checks: Compare the output from VLIST and VXREF, for example. The first reports variables actually used, the second lists all names imbeded in the program. Names reported by VXREF but not by VLIST ought to be examined. On the one hand, that area of code was never executed (but maybe it should have been). Another possibility: In the case, for example, OOPS is listed by VXREF, but not by VLIST, check those lines. If you see something like IF OOPS=15 THEN, then the question arises, was OOPS ever assigned a value in the first place?

|Limitations

These tools are extremely version-sensitive--currently coded to suit GWBASIC.EXE Version 3.23, specifically. They "read" a program as it is stored in memory, in so called tokenized form. Although most of the tokens and other internal codes and data formats have remained much the same over the last several years, working storage areas used by the interpreter tend to "move" with each successive release and new versions. Thus: If these tools do not work correctly it is most likely because of internal addressing differences, i.e., the interpreter in question is not version 3.23.

All of these programs were written very compactly to cause the least space impact while merged-in with an end product program under development. The coding style they exhibit is not meant to be pretty; they are dense so as to achieve optimum performance during their use.

Their "crash handling" logic is equally simplistic: Let the interpreter take care of it. If ERR=7 or ERR=9, design capacity limits have been exceeded.

See the next page for some specific limits and "exceptions" ....

|

tool	static size	run space needed
¨LHITS	839 bytes	8100 bytes
¨LXREF	823	8100
¨VFIND	939	<100
¨VLIST	690	<100
¨VXREF	1775	3500 plus 3 for each unique name

LHITS and LXREF provide for a maximum of 999 references. ERR=9 if more than that are encountered.

VXREF works for up to 255 unique names. ERR=9 if more are found.

If LHITS or LXREF or VXREF run out of memory, ERR=7 will be reported.

VFIND treats "untokenized" key words as if they were variables, thus, SEG can be found as if it were a variable name. Also, searches for names B or BF will also produce line numbers containing LINE statements that use these argument "codes".

VLIST uses an array called O!--it too will be shown in the output report (as the last one named if your program does not have a similarly named one).

(continued)

|

All of these tools suppose the program being analyzed is free of simple syntax errors. If a tool hangs, or crashes, or produces bad output it may well be because it bumped into unexpected "junk".

Yes, it would be possible to "improve" them so they would not suffer from the limitations listed above. But, they would be bigger. And slower. And a lot more complicated. So: They are the way they are on purpose. We use them daily in our shop just as you see them here. They work quite well for us because some of our conventions obviate their limitations ....

No program of ours ever has more than 255 unique variables (thirty or forty is typical of our largest ones).

No program ever "branches" to more than 999 different lines (we do not write spaghetti-code).

We never use O by itself as the name of a variable (to preclude eye-ball confusion with a zero).

No program of ours uses line numbers ranging anywhere near 9000 (they are always numbered starting at 1000 and incremented by 10). Obviously, you can renumber your tools to start higher, if you want to.

|LHITS

LHITS --- Line Hits

This tool will provide a list of line numbers that have been branched-to, and which lines referenced them, after a program has been run to some point.

```
Example:  1130  1200, 2020, 3130

Read as:  Line 1130 was referenced from within lines 1200, 2020, and 3130
          by a GOTO, GOSUB, THEN, ELSE, etc.
```

To use LHITS merge it into your program. RUN your program. After an END, STOP, or Ctrl-Break, do a GOTO 9000.

It is also possible to simply insert a GOTO 9000 at a strategic place in your program to invoke the tool automatically.

Notice this is a dynamic (run-time-live) tool. Compare with ¨LXREF which is a static code analyzer.

|LXREF

LXREF --- Lines Cross Reference

This tool will provide a list of line numbers that are branched-to, and which lines contain those references.

```
Example:  1130  1200, 2020, 3130

Read as:  Line 1130 is referenced from within lines 1200, 2020, and 3130
          by a GOTO, GOSUB, THEN, ELSE, etc.
```

To use LXREF merge it into your program then type RUN 9000.

Be prepared to hit <Pause>--scrolling may occur if output is sent to the monitor. (Only three PRINT statements need to be changed to send output to the system printer; they are at the very end of the tool program; LPRINT is an easy fix when you prefer this.)

Notice this is a static code analyzer. Compare with ¨LHITS which is a dynamic tool--it only reports lines actually branched-to after a program has been run to some point.

|VFIND

VFIND --- Variable Find

This tool searches a program in memory and reports the line numbers of any line that contains a particular variable's name.

```
Example:  Find X4

Output:   2020  2040  3650

Read as:  X4 is contained one or more times in each of the lines listed.
```

To use VFIND merge it into your program then type RUN 9000. After the prompt "Find", type the name of the variable to be searched for.

Notes: Lower case letters typed after Find will automatically be upshifted.
Exact spellings are important; X4 is different than X4$.
Array names should be typed like IR(--with left parenthesis only.
To find function names include the FN. Like FNC or FNC(or FNX$.
Also, see ¨Limitations about the special cases of names B and BF.

|VLIST

VLIST --- Variables List

This program prints the contents of the variables index as it has been constructed by the interpreter--the names are ordered as they were added to the index since the last RUN was done.

To use VLIST merge it into your program. Run your program and exercise most of its routines. After an END, or STOP, or Ctrl-Break, type GOTO 9000.

It is also possible to simply insert a GOTO 9000 at a strategic place in your program to invoke the tool automatically.

Notice this is a dynamic (run-time-live) tool. Compare with ¨VXREF which is a static code analyzer.

Output from this tool reflects the data type of each variable, regardless of whether or not those appendages ($%!#) were included within program statements themselves. Also, this program uses an array called O!(--if your program did not use this name, this one will be listed as the last one added to the index.

|VXREF

VXREF --- Variables Cross Reference

This program prints an alphabetized listing of all variables named in a program. Each name is followed by the line numbers of lines containing one or more of those references.

To use VXREF merge it into your program then type RUN 9000.

Be prepared to hit <Pause>--scrolling may occur if output is sent to the monitor. (If you prefer printer output, scan the program for PRINT commands and change them to LPRINT.)

Notice this is a static code analyzer. Compare with ¨VLIST which is a dynamic tool--it only reports the names of variables actually encountered after a RUN has been done to some point. (And that tool does not list any line numbers.)

Also: This program treats names like X4 and X4$ as distinct, based on how they are stated in programming expressions. Whereas they may in fact be treated the same by the interpreter -- and ¨VLIST -- as would be the case if DEFSTR X was encountered when the program in question is executed.

|GeeWhiz

```
+-----------------------------------------------+ ¦¦¦¦¦¦¦¦¦¦¦
¦ Programming in GW-BASIC² on a PC compatible?  ¦ ¦¦¦¦¦¦¦¦¦¦¦
+---+                                           +---+ ¦¦¦¦¦¦¦
¦¦¦ ¦ Tired of trying to find help in the manuals?  ¦ ¦¦¦¦¦¦¦
¦¦¦ +---+                                           +---+ ¦¦¦
¦¦¦¦¦¦¦ ¦ Tired of trying to figure out what they mean? ¦ ¦¦¦
¦¦¦¦¦¦¦ +---+                                           +---+
¦¦¦¦¦¦¦¦¦¦¦ ¦ Tired of having to learn via trial and error? ¦
¦¦¦¦¦¦¦¦¦¦¦ +-----------------------------------------------+

------------- ¨WhizWare has a solution, gee whiz. -----------

   We call it GeeWhiz -- An on-line language information
   system for anyone writing programs in GW-BASIC.

      GeeWhiz is quick.  Easy to install and easy to use!
```

And it runs on nearly any PC running DOS 3.0 or later. Color or mono, with or without a Mouse. Only half a meg or so of system disk space is needed to hold all of this compacted wealth of knowledge.

(continued)

The "on-line" concept: While editing and debugging you can SHELL to GeeWhiz, find facts quickly and return to where you were. Done with a Function Key, we call this a hot-key, hot-line for spontaneous programming help.

GeeWhiz is loaded with proven tricks of the trade, too. It was written by a professional programmer that has written thousands of successful applications in BASIC, dating from the earliest versions of CBASIC, MBASIC and the many variants since. Thus, the "techniques" examples are really useful modules of code that really do work.

Look at ¨WhizWare for how to contacat us.

---

[2] GW-BASIC is a registered trademark of Microsoft Corporation