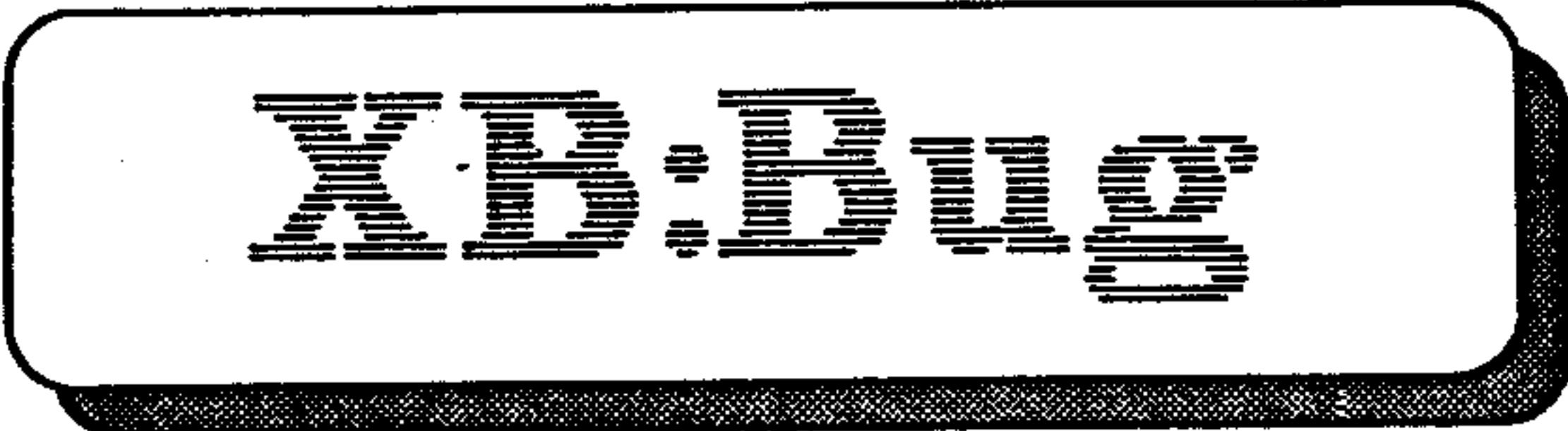


## THE NEXT STEPS

When you have finished with these examples, select a short program of your own. Make sure that the debug program is loaded as before, load your own program, and experiment with the different functions available to you. Remember that you can get a reminder of all the functions by entering ? from the debugger prompt.

You will find the debugger an invaluable tool when you are writing programs. Keep it loaded in low memory (unless you are using the low memory space for assembly) at all times when you are writing an Extended BASIC program. Whenever you encounter a problem and your newly written code does not behave as you expect it to, it gives you a powerful tool to track down the source of the problem. The more you use it, the more you will find it can do for you.

Genial Computerware  
Box 183  
Grafton, MA 01519



XB:Bug

©1987 J. Peter Hoddie

Distributed by Genial Computerware

## ENTERING THE DEBUGGER

To call up the debugger press then SHIFT and CTRL keys simultaneously. You may also enter the debugger with a CALL LINK("GOBUG"). In either case, you will be presented with the main debugger prompt (->). The following commands will work at this prompt, simply by pressing the first letter in their name.

## USING XB:BUG WITH XB/AL PROGRAMS

XB:BUG takes up about 5K of memory. If you include assembly with your Extended BASIC program, which is becoming common these days, you have a decision as to which version of XB:BUG to use. The BUG program loader of XB:BUG puts the program in low memory and you can load your assembly code (if it isn't too long) on top of XB:BUG. If you have already loaded your assembly code, you can use the DIS/FIX 80 file BUG/REL which will load after your assembly code if there is enough room. The final alternative is to use the DIS/FIX 80 file BUG/A000 which loads XB:BUG entirely into high memory which means you have the entire low memory expansion for assembly code as usual. However, if you use BUG/A000, your BASIC program, when running, can not leave fewer than 6000 bytes of program space free.

## COMMAND SUMMARY

### ARRAY

This allows you to inspect the contents of an array. The array that will be inspected is the last array that was listed using the Variables command. You will be prompted for the "Parameters" of the array. If you wished to look at element (3,14) of an array you would enter 3 for "#1 Parameter" and 14 for "#2 Parameter." If you look up in a string array and the string is a null string (length of zero), a blank line will be displayed.

## BREAKPOINTS

Displays current breakpoint or break-range and allows you to modify these values. To turn the break point off enter -1 for the first value and enter as the second. To set a single line breakpoint enter the line number at the first prompt and again at the second. To set a break-range, enter the low line number at the first prompt, and the high line number at the second.

Note that you must turn off the breakpoint if the debugger is called in this manner, or the debugger will continuously enable itself.

*Note:* The reason for allowing a Break-range is because some lines may execute too quickly for XB:BUG to be able to enable itself. A Break-range should generally be used rather than a single line Break-point. If you really need a single line break point, put a CALL LINK("GOBUG") on the line and you will automatically enter XB:Bug.

*Warning:* Do not set a break point while in command mode, only set it after you are sure the program has started executing. If you set a break point while in command mode, the debugger may enable itself while Extended BASIC is executing Pre-Scan.

## CHANGE

This allows you to change the value of any numeric variable. To use it you must first look the variable up using the V or A command. The last numeric variable value displayed using the Variables or Array command is the value that will be changed. When you select Change, the current value will be displayed and you will be prompted for a new value. To keep the same value, just hit enter. If you enter text, a value of 0 (zero) will be assigned to the variable, so it is a good idea to check that the variable was changed correctly.

## DATA/READ

Gives the line number from which the next READ will get its DATA and also shows the next actual DATA item that will be read.

## FILES

Lists the unit number and device name associated with each open file. The "mode" that the file was opened in is also given which is either INPUT, UPDATE, APPEND, or OUTPUT. If there is any data in the input/output buffer of this file it will be displayed. The buffer could contain data that was read from the device, that will be written, or that has been written. There is no way to tell, except common sense, into which category the data falls. Note that this is an ASCII dump of the data buffer. If the buffer contains floating point numeric data it will appear as garbage.

## GRAPHICS

This section gives information on 3 main graphics categories: 1) Character definitions, 2) Color definitions, and 3) Sprite status. You will be presented with a menu of the 3 above items. After making your selection you will be asked for the specific character definitions, color tables, or sprites that you wish to see. If you want to see only one, enter its number at the "FROM" prompt and hit enter at the "TO" prompt. To see a range enter the beginning of the range at the "FROM" prompt, and the end of the range at the "TO" prompt. When the display is complete, you will be returned to the main prompt.

For Characters the valid range of definitions is 30 to 143. For Colors it is 0 to 14. For Sprites it is 1 to 28. If a Sprite is not active (deleted) it will be listed with a character of 160.

## KILL SOUND

This turns off the sound chip. When you return to BASIC, the next sound will follow, after the current sound (which is now silent) has finished. That is to say, if you interrupt a 4 second tone, after it has been sounding for 2 seconds, and it is followed by another tone, it will take 2 seconds, after you exit the debugger, before the second tone will start. So in effect the Kill Sound silences the current tone, but does not stop its timer.

## LIST

Lists lines. You will be prompted for the start and stop lines. If you wish to list only one line, enter that number as the first line number and hit ENTER for the second one. If there are any Breakpoints set (by the BASIC command BREAK, not XB:BUG's B command), the line number will be preceded with an asterisk. Note: the line listing format is not exactly the same as is used by BASIC but it is workable. NOTE: This command uses the MATCH function described below.

## OTHER VARIABLE SPACE

This command allows you to look at the variables from the main program or other subprograms regardless of where you entered the debugger. After you select Other you will be presented with a list of 3 choices. 1 is for first, which means to return to the variable space that was active when you first entered the debugger. 2 is for subprogram, which means to switch to the variable space of the last BASIC subprogram looked up using the S command. 3 is for user defined variable space, which is defined by executing a CALL LINK("SETSYM") in the your program. To clarify, say you enter XB:BUG from the main program, but want to look at a variable in the subprogram FRED. Use the S command with a Match string of FRED to allow XB:BUG to look up the subprogram FRED. Now select the O command with choice 2, subprogram, to switch to the variable space of the subprogram FRED. Now if you hit V, the variables that are listed are those of FRED, not of the main program. To switch back to the variable space of the main program, simply select O again, with choice 1, first variable space. The main reason for the presence of the CALL LINK("SETSYM") command is so that you can keep track of the main program variable space, so that if you enter XB:BUG from within a subprogram you can still look at the variables of the main program. In general, if you are using subprograms you should make the CALL LINK one of the first lines of your program to insure that you can always access the main programs variables by using the O command with option 3, user defined variable space. The O command has been included primarily for advanced users. For an example of its use see the example section of this manual.

## PROGRAM

This supplies general information about the program including line number executing, the ON ERROR line number if one has been set, and Option Base. If the following are active, you are notified: On Break, Trace.

## QUIT

Exits the debugger and returns you to whatever was going on when the debugger was entered.

## SUBPROGRAMS

Lists all defined subprograms. If the subprogram is written in BASIC the line number which contains its SUB statement will be listed after the words "AT LINE." If the BASIC subprogram is currently active, this is indicated by "IN USE". If BASIC is executing a Subprogram, only the subprogram calls that are valid from within that subprogram are displayed. Note: This command uses the MATCH function described below.

## TRACE

This feature traces back all pending GOSUB and SUBPROGRAM returns. They are listed in order that they are pending, i.e. the first listing is the next pending return. A gosub entry tells FROM what line the GOSUB was executed and TO where it went. It will return to the line listed after FROM. For example, the statement

```
10 GOSUB 300
```

would result in a listing of

```
FROM 10 TO 300
```

from the Trace command.

A subprogram listing gives the name of the subprogram and the line to which it will return when it reaches a SUBEND or SUBEXIT.

*Note:* Because of the way the stack is maintained by BASIC, it is possible, but very unlikely, that a completely absurd entry may appear.

## VARIABLES

List variables (and functions) with their current values. In the case of an array, the dimensions are given. If the program is in a subprogram, the variables for the subprogram are shown. *Note:* This command uses the MATCH function described below.

?

This displays all valid keypresses. Sort of a primitive "help" key. Only here because it only cost 5 bytes!

## MATH FUNCTIONS

You can perform simple math calculations from the debugger. At the main prompt hit the function you wish to perform: +, -, \*, or /. You will then be prompted for two numbers and the result will be displayed. Please note that these numbers may be floating point, and should be entered in the same format as numbers for BASIC.

## MATCH FUNCTION

Several of the commands in the debugger prompt for a MATCH string. This is a method for looking up specific data in a long list of information. If you want the entire list, just hit ENTER at this prompt and all data will be presented. If you want to look up the value of the variable A, you can just enter A= for the Match string and it would only display the variable A.

The Match command also supports "wild cards." These are special characters that allow you to do more complicated searches. The first "wild card" character is the question mark (?). This tells the search routine to skip to the next character. So if you wanted to search for all 2 character variable names you could enter ??= as the match string. If you wanted to see all 2 character variable names beginning with the letter Z you could enter Z?= for the match string.

The other "wild card" character is the asterisk (\*) which says find the string after me. For example, the Match string \*Z says find any string that has a Z. The Match string A\* says find any string that begins with A. If you wanted to search for all variables that began with the letters FLAG, you would select a Match string of FLAG\* and that would do it. A "?" is treated like any other character after a "\*" and you may only have one "\*" in a Match string. Thus to search for a string containing a "?" use the Match string \*?. Wild cards are a very powerful tool for searches, as you will discover in using them.

## NOTES

If a command has nothing to display, it shows nothing and returns you to the prompt.

If you ask for PROGRAM information or DATA/READ and a program is not active, the information will probably be garbage.

Just about any output can be paused by holding down the space bar.

XB:BUG will not work with Myarc XB II because of the radically different layout of memory. It will work with any version of Extended BASIC based on TI's Extended BASIC including Mechatronics XB II+, and MG's GK Extended BASIC.

## THANK YOU

This program owes much to Barry Traver, who constantly reported bugs and quirks along with many helpful suggestions. XB:BUG was inspired by and is dedicated to Jim Peterson and Barry Traver who manage to push Extended BASIC to its limits. It is hoped that this program will make their jobs a little easier. Thanks to Todd Kaplan for creating ALSAVE which allowed me to create the fast loading BUG file. Thanks to Mike Dodd for his helpful last-minute suggestions. A special thanks to Doug Warren and Craig Miller of Millers Graphics for creating Explorer, an assembly language debugging tool (and much much more) which was invaluable in creating XB:BUG. A quiet thanks to MS for his Xerox machine.

## EXAMPLES

### EXAMPLE 1

1. From Extended BASIC, load the debugger by entering  
RUN "DSK1.BUG"
2. Load the demo program by entering  
OLD "DSK1.DEMO1"
3. Just to see what the Extended BASIC program is type  
LIST  
and then type  
RUN  
and let the program finish so we can continue with the next step.
4. When the cursor re-appears enable the debugger by hitting SHIFT and CTRL at the same time. You will be presented with its main prompt and we are ready to begin.
5. First list the program from the debugger, using its list command. Hit L, then at "FROM" enter 10 and at "TO" enter 1000. Hit ENTER at "MATCH" and the program will be listed for you. Note that this listing format is rather different from that used by Extended BASIC, in that it is a bit more spaced out.
6. Now lets try out the match function. Hit L again and enter 10 and 1000 again, but for the match string enter "DISPLAY". Only the lines that contain the word "DISPLAY" are listed because of the use of the wildcard character "\*" described under the Match function section.

7. Now try out the Variables command. Just hit "V" from the main prompt and hit ENTER for the Match string to see a list of all variables and their values. Note that the dimension A\$ is just listed with its dimension, not its data. The value of DELAY is 151, not 150, because the FOR-NEXT loop increments DELAY before comparing to the "end" value of 150 and deciding to end the loop.

8. To see the value of a particular element of the array A\$, hit A for Array and enter 6 for "#1 Parameter" to see A\$(6). You can look at other elements of A\$ the same way.

9. Now look up the value of the variable Z using the Variable command. Hit V for Variable and enter Z= as the Match string. This will display Z=8.

10. To change the value of Z, which we just looked up, hit C for Change. You will see the current value which is 8. At the "TO" prompt enter a new value of 12.3. You can just hit enter to keep the current value.

11. Hit V and hit ENTER for the Match string to see the changed value of Z.

12. Now hit S to get a listing of all the Subprograms that are available. In this case the only one is CLEAR.

13. Now hit D for Data to see what the next item that would be read by the READ statement is. It will be at line 1000 and the data will be HAROLD.

14. To see that this is correct, hit L to list, enter 1000 at "FROM" and hit ENTER at "TO" and line 1000 will be listed. This contains the DATA statement. Note that there are 8 data items, the eighth of which is HAROLD, but we only read 7, so the data displayed at step 13 is correct.

15. Hit Q to exit the debugger and return to BASIC.

16. From BASIC type  
PRINT Z

to see that indeed Z now equals 12.3 as we set it in step 10.

## EXAMPLE 2

1. If you have just done Example 1, the debugger is already loaded, so you can proceed to step 2. Otherwise load the debugger by entering  
RUN "DSK1.BUG"  
from Extended BASIC.
2. Load and run the demo program by entering  
RUN "DSK1.DEMO2"
3. When you hear a tone, enable the debugger by hitting SHIFT-CTRL.
4. By now you have probably gotten quite annoyed with the noise coming from your monitor so hit K to Kill sound.
5. Now List the program to see what we are working with. Hit L and then 10 for "FROM" and 100 at "TO." Note that the program essentially just clears the screen, defines character 128, sets color set 12 to 4,3, and enables 10 sprites.
6. Hit G for Graphics, and select 1 for Characters. Enter 128 for "FROM" and hit ENTER for "TO." You will see the definition of character 128 as it was set in line 20. Note that the character itself is shown after the definition.
7. Hit G again and select 1 for Characters. Enter 32 for "FROM" and 42 for "TO." This will list all characters from 32 to 42 with their definitions.
8. Now we'll look at the color sets. Hit G and select 2 for Colors. Enter 10 for "FROM" and 14 to "TO." This will show you the color sets from 10 to 14 as they are defined in BASIC. The format is: Color Set Number, Foreground color, Background color. Also note that Extended BASIC default colors are foreground of 2 (black) and background of 1 (transparent).
9. Let's look at the sprite data next. Hit G and select 3 for Sprites. Enter 1 for "FROM" and 11 for "TO." This will show you data on sprites 1 to 11 in the same format as a CALL SPRITE statement, which is: Sprite Number, Sprite Character, Sprite Color, X position, Y position, X velocity, Y velocity. Note that if a sprite is deleted with the DELSPRITE command or was never created, it will have a character of 160.

10. Hit P for Program information to see what line is currently being executed, the Option Base (which defaults to 0), and that ON ERROR is set to line 70 as we set it in line 5.

11. Hit Q to exit the debugger and return to the running program. When you are bored with the dull demo, just hit FCTN-4 to stop it.

## EXAMPLE 3

1. Make sure the debugger is loaded as in step 1 of example 1.
2. Load and run the program for this example by typing  
RUN "DSK1.DEMO3"
3. You will be prompted to "HIT ENTER." Don't do it. Instead enable the debugger by hitting SHIFT-CTRL.
4. From the debugger hit S to list Subprograms and hit ENTER for the Match string. You will see a listing that says  
FRANK AT LINE 500 \*IN USE\*  
This means that there is a subprogram named FRANK that begins at line 500 and that is currently being used.
5. If you now list the program with the L command, FROM 5, TO 2000 and ENTER for Match string you will see the program. Essentially the program does a GOSUB to 500 which CALLS FRANK and the program sits around doing nothing at line 20.
6. Now hit T for TRACE. This will tell you that FRANK RETURNS TO 500 and that there is a GOSUB that has not returned that was executed at line 10 and went to line 500. This is obviously the GOSUB 500 in line 10 that will be resolved by the RETURN at line 510.
7. Now hit V for variables and hit ENTER for Match string. You will only see the variable A\$ listed, not the variable X from line 5 because the Variable command only shows variables that are valid for the subprogram you are in.
8. Now we will set a Breakpoint at line 20, so that when the program gets there, the debugger will appear automatically, without having to hit a key. To do this hit B for Breakpoint and enter 20 for FROM and 20 for TO.
9. Exit the debugger with a Q to return to the program.

10. Hit enter to allow the program to continue.
11. When the program reaches line 20, the debugger will appear. First hit P for program information to see that, in fact, it is at line 20.
12. Now remove the break point by hitting B, and entering a value of -1 for FROM and hit enter for TO.
13. Hit V to list the variables. Note that now X is present since we are no longer in the subprogram, but that A\$ is gone because it only exists inside the subprogram FRANK.
14. Hit Q to exit the debugger and FCTN-4 to end this rather dull demo.

#### EXAMPLE 4

1. Make sure that the debugger is loaded as in step 1 of of example 1.
2. Load the program by entering  
RUN "DSK1.DEMO4"  
from Extended BASIC.
3. The program will start. It is a very simple program that opens up a file called "TEXT" on DSK1 and displays the data on the screen, with a delay between lines. Before the program is done running, enable the debugger by hitting SHIFT-CTRL and list it by hitting L, enter 10 for "FROM", 100 for "TO", and hit ENTER for Match string.
4. The whole point of this example is to show the File command, so next hit F. It will show you that unit #1 is the file "DSK1.TEXT" and was opened in INPUT MODE. On the line after the word BUFFER you will see the current contents of the input buffer.
5. Exit the debugger by hitting Q.
6. Before the program ends, try steps 3 through 5 a number of times to see how the buffer changes.

#### EXAMPLE 5

1. Make sure that the debugger is loaded as in step 1 of of example 1.
2. Load the program by entering  
RUN "DSK1.DEMO5"  
from Extended BASIC.
3. Wait about 5 seconds and then enter the debugger by hitting SHIFT-CTRL. List the program using the List command and lines 1 to 500. Note that the program does a CALL LINK("SETSYM") at first to define the variable space of the main program (see the O command for details), then assigns some variables in the main program and then assigns some variables in subprograms, before coming to a complete halt at line 230.
4. At this point, hit V and and ENTER for match string, to see the variable of the subprogram FRED where the program is currently executing.
5. Now we will look at the variables of subprogram JOE. Hit S and enter JOE as a Match string. Now hit O and select 2 to switch to the variable space of the last subprogram looked up with the S command, namely JOE. Now hit V and ENTER for Match string to see the variables of the subprogram JOE.
6. To see the variables of the main program hit O and select 3 to switch to the user defined variable space, which in this case is the main program as defined by the placement of the CALL LINK("SETSYM") statement. Hit V and ENTER as Match string to see these variables.
7. To get back to the variable space of FRED, we can either look it up using the S command as in step 5, or simply hit O and select 1, to switch to the variable space that we entered the debugger in, which in this case is JOE. Now hit V and enter for Match string to see that we are back in the variable space of FRED.
8. Hit Q to exit the debugger and FCTN-4 to end the program. You might want to try creating some of your own programs like this to learn how to use the O command.