For MacsBug 6.2

MacsBug 6.2
Software Release Notes

Developer Technical Publications

MacsBug 6.2 Software Release Notes

MacsBug 6.2 Quick Reference

This section provides a summary of changes and additions to MacsBug.

Command/Trap/ Variable	**Syntax**	**Effect/Change**
Command-D	Command-D	Displays a menu of procedure names from which you can select a name to insert in the command line. This is not a new command; it is a way of implementing the Command-: command on German and Scandinavian keyboards.
Command-B	Command-B	Scrolls the command line buffer up.
Find command	F *addr nbytes expr* | "*string*"	Returns the address where it finds the specified pattern.
	F[B|W|L|P] *addr nbytes expr*	Returns the address where it finds the specified byte, word, long word, or pointer.
ATP command	ATP	If ATR is off, the ATP command plays back information from the most recent ATR.
GT command	GT *addr* [';*cmds*']	Breaks at the specified address and executes one or more commands.
DebugStr trap	DebugStr ("*string* [; *cmd*]...")	Following a break set with the Debugger trap from within the source program, it displays *string* and executes one or more commands.
Printf dcmd	printf "*format*" *arg* ...	Formatted output command.
TargetZone variable	TargetZone	Specifies the zone currently set with the HX command.

MacsBug 6.2: What's New, Improved and Documented

This note describes the software changes and additions that make up the release of MacsBug version 6.2. These changes and additions have basically one aim, to make MacsBug an easier tool for you to use as you hunt for bugs and test code.

One of the major changes to MacsBug is that it now works reliably with all Apple monitors and all third-party monitors if their slot ROM and driver software have been designed according to the guidelines presented in *Designing Cards and Drivers for the Macintosh II and Macintosh SE.* MacsBug 6.2 also runs under A/UX. Please read the section "Using MacsBug under A/UX" for additional information. In addition to its increased portability, MacsBug 6.2 includes

- A new display that shows you the name of the current application, the memory management scheme (24 bit/32 bit) currently used, and whether you have access to virtual memory.
- New options for the Find command that allow you to specify the width of the pattern for which MacsBug searches. You can even use one of these options to have MacsBug look for pointers.
- A more detailed heap dump display.
- An extension to the GT command that allows you to specify one or more MacsBug commands to be executed once the specified breakpoint is reached.
- A standard printf dcmd (debugger command) that allows you to produce formatted output.

In addition to these specific changes, you should find MacsBug 6.2 easier to use. Whenever possible its output has been made more articulate and its displays more intelligible.

This software release note also describes an extension to the DebugStr trap, which was introduced (but not documented) with the 6.1 software release of MacsBug.

Please read through the following pages for more detailed descriptions of the items listed above. You should try out the new commands and command options whenever possible. When you are familiar with the new material and are ready to work with MacsBug 6.2, you might want simply to consult the "MacsBug Quick Reference" section on the opposing page, or use the on-line help to jog your memory.

New MacsBug Display Information

MacsBug 6.2 has changed the display to provide the following information:

- The name of the application using the processor when MacsBug was invoked
- Whether the machine is in 24-bit or 32-bit memory mode
- Whether virtual memory was running when MacsBug was invoked

Figure 1 shows the new MacsBug display. The new information is shown in the status region of the MacBug Display, in the area between the stack and the register information.

The Current Application Name: MacsBug 6.2 displays the name of the application using the cpu when MacsBug is invoked. In Figure 1, the current application is Finder.

If an unexpected crash lands you in MacsBug, check this item first. If your application is running in a multitasking environment (Multifinder) it is possible that one of the background applications has caused the crash. If this is the case, the name of the current application shown in theMacsBug display will not be the name of the foreground application.

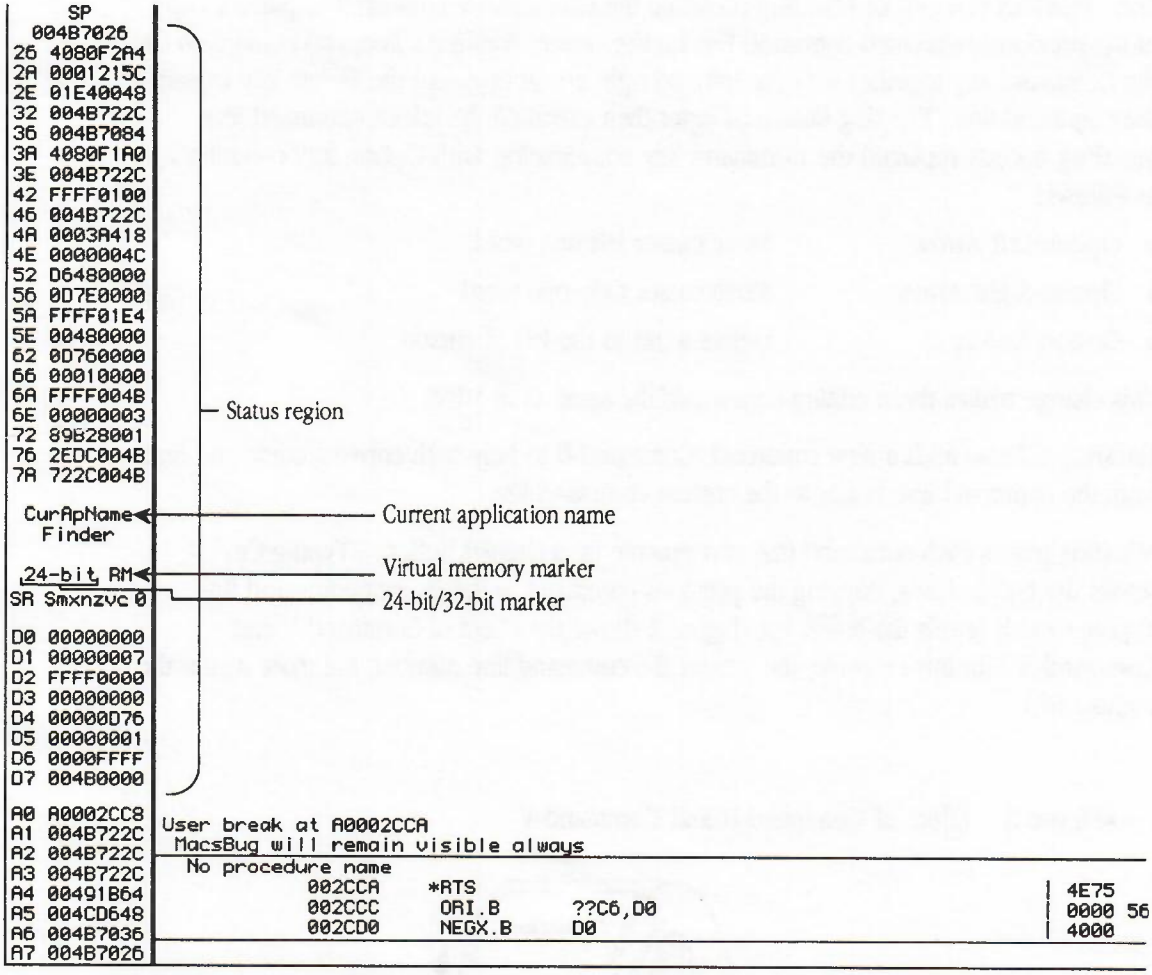
24-bit and 32-bit Marker: MacsBug 6.2 displays the message "24-bit" or "32-bit" to indicate whether the machine is in 24-bit mode or 32-bit mode.

Being aware of which mode the machine is in could save you time and trouble in identifying the cause of some bus errors (in MacsBug the error would be "Unable to access that address."). For example, to implement drivers that access a NuBus board you need to switch the hardware to 32-bit addressing mode. If you now de-reference an address that uses the high byte to hold data, you will get a bus error because the address does not refer to a valid location. It is your responsiblity to make sure that your program is not using invalid addresses either by switching back to 24-bit mode or by using the Strip Address routine to strip the misleading high byte from the address. For additional information see "Compatibility: Rules of the Road" in the January 1990 issue of *Develop,* and "The Memory Manager" in Volume II of *Inside Macintosh.*

In short, the marker tells you what memory management scheme the machine is using. It is then up to you to make sure that your application is behaving appropriately given that state.

Virtual Memory Marker: MacsBug 6.2 displays one of three codes to indicate whether virtual memory is being used and whether MacsBug can rely on the Memory Manager to swap pages if the program being debugged makes use of virtual memory.

■ **Figure 1** MacsBug 6.2 Display



The codes displayed have the following meaning:

- RM — virtual memory is not being used (Real Memory).
- VM — memory manager can swap pages if MacsBug requires it.
- vM — MacsBug was invoked while the memory manager was swapping pages; now the Memory Manager cannot swap pages for MacsBug.

Command Line Editing Commands

Using MacsBug often involves having to type complicated expressions on the command line. Previous versions of MacsBug provided the Command-V command to place a copy of the previously executed command line on the current command line; you could then use the Command key together with the left and right arrow keys and the Delete key to edit the command line. Pressing Return or Enter then executed the edited command line. MacsBug 6.2 has replaced the Command key combination with Option key combinations as follows:

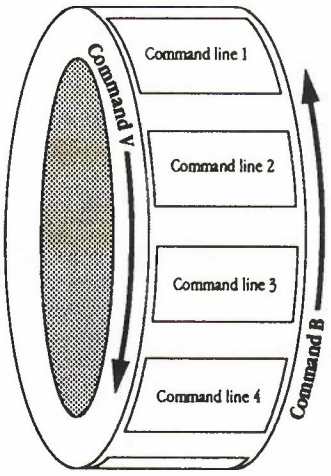
- Option-Left Arrow — Move cursor left one word.
- Option-Right Arrow — Move cursor right one word
- Option-Delete — Delete word to the left of cursor

This change makes these editing commands the same as in MPW.

MacsBug 6.2 also adds a new command, Command-B to help with copying command lines from the command line buffer to the current command line.

MacsBug places each command that you execute in a circular buffer . Typing Cmd-V, scrolls the buffer down, copying the previous command to the current command line. Typing Cmd-B scrolls the buffer up. Figure 2 shows the effect of Command-V and Command-B. (In this example, the greater the command line number, the more recent the command.)

Figure 2 Effect of Command-B and Command-V



The following example shows you the process of entering and editing commands using Command-B and Command-V. If you have the time, you should try duplicating these steps. Learning how to use these two commands can make your work with MacsBug much easier.

The series of boxes below show the current command line, the command line buffer and the copy pointer. The position of the copy pointer determines which line in the command buffer will be copied to the current command line. Pressing Command B, Command V, or Return changes the position of the copy pointer. The text to the left of the boxes explains what MacsBug does as the user types and enters commands.

MacsBug has just been invoked. The command line is empty; the command line buffer is also empty.

The user types vol.

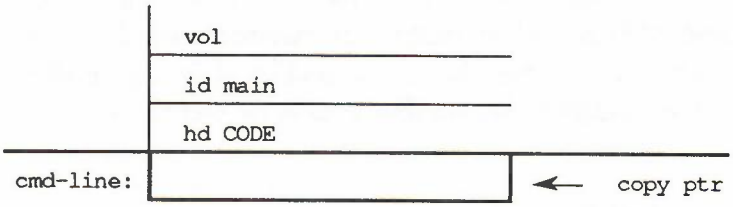
The user presses Return. MacsBug executes the command. It displays the output for the command in the output region, adds the command to the command buffer and clears the command line for new input.

The user types id main.

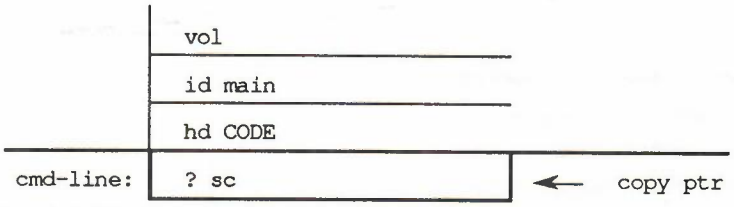
The user presses Return. MacsBug executes the command, adds it to the command line buffer, and clears the command line for new input.

The user types hd CODE.

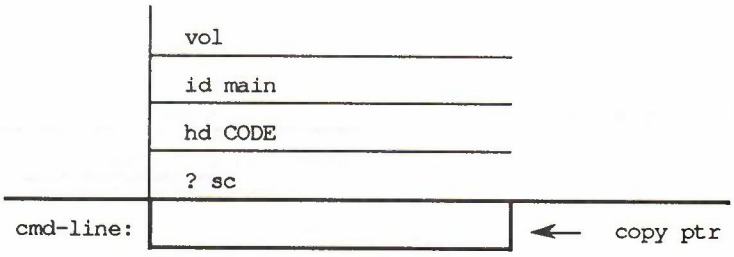
The user presses Return. MacsBug executes the command and adds it to the command line buffer. The command line is clear and ready for input.



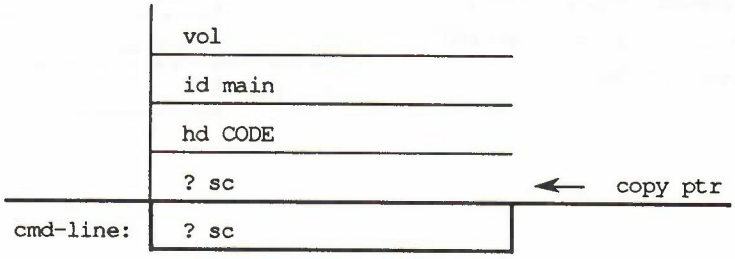
The user types ? sc.



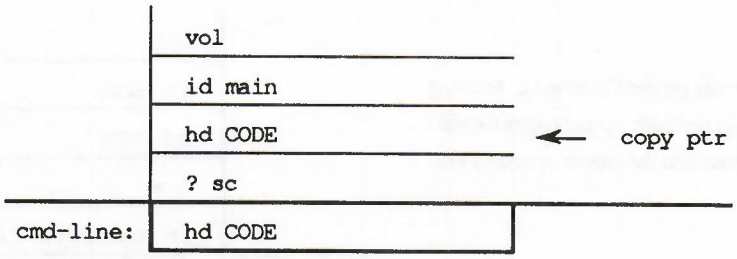
The user presses Return. MacsBug executes the command and adds it to the command line buffer. The command line is clear and ready for input.



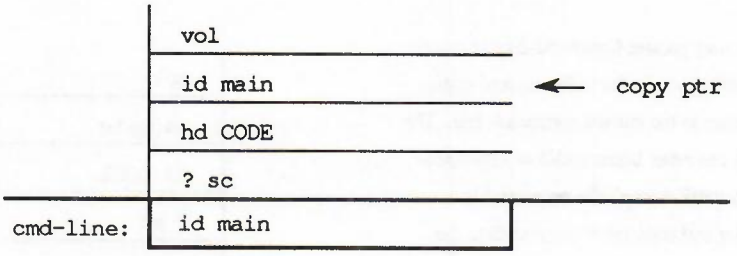
The user enters Command-V. MacsBug scrolls the command line buffer down one position and copies the command line at that position to the current command line. Note that the arrow indicates a new position for the copy pointer.



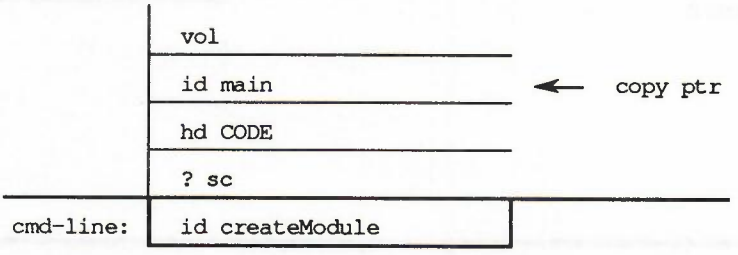
The user enters Command-V. MacsBug scrolls the command line buffer down one position further and copies the command line at the new position to the current command line, replacing the contents of the current commandline.



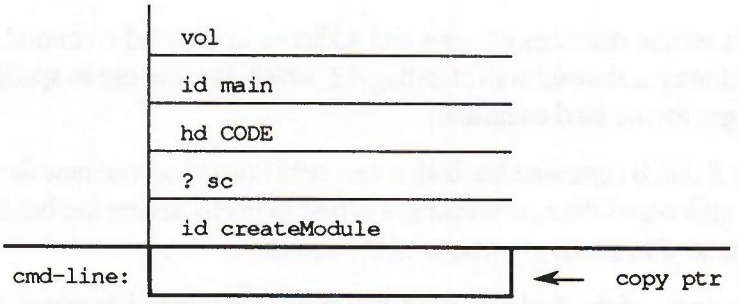
The user enters Command-V once more. MacsBug scrolls the command line buffer down and copies the command at the new current position to the current command line.



The user edits the current command line by backspacing four times to delete main and then types createModule.

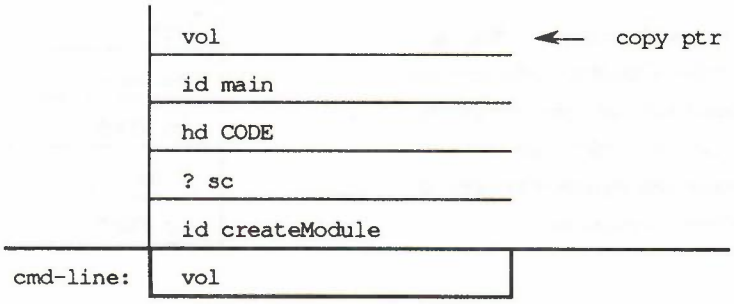


The user presses Return. MacsBug executes the id createModule command, adds it to the command line buffer, and restores the buffer to its default position.

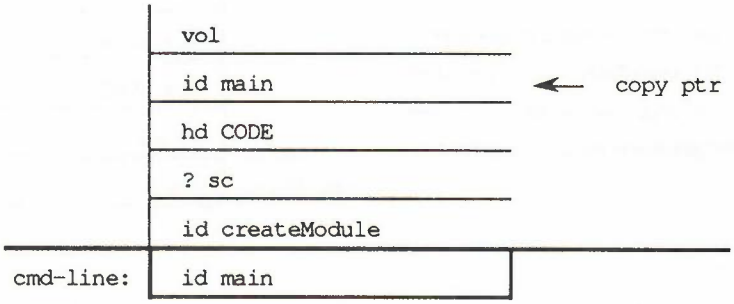


If the user pressed Command-V now, MacsBug would copy the id createModule command to the current command line. If the user pressed Command-B instead, MacsBug would copy the vol command to the current command line.

The user presses Command-B. MacsBug scrolls the buffer up and copies the vol command to the current command line.



The user presses Command-B once more. MacsBug scrolls the buffer up and copies id main to the current command line. The user can enter Command-B or Command-V to scroll through the command line buffer and copy other commands to the command line, edit the command on the current command line, or press Return to execute it.



The Find Command

This section describes changes and additions to the Find command. It also describes the Find macros shipped with MacsBug 6.2, which you can use to specify one of several ranges for the Find command.

The F (Find) command has both a new behavior and several new flavors in MacsBug 6.2. To understand the new features, it is best to briefly review the behavior of the Find command in previous versions of the software.

The syntax of the Find command, which has not changed, is shown on the next page:

F *addr nbytes expr* | '*string*'

addr specifies the starting point of the range where MacsBug should begin the search . MacsBug uses the value of *addr* + *nbytes* - 1 to determine the end point of the range.

expr specifies the value to search for.

'*string*' specifies the string to search for.

In MacsBug 6.1, if you entered the command

```
F 0 100000 1234
```

it might return the following:

```
Searching for 1234 from 00000000 to 000FFFFF
  000044B9  1234 0161 0000 D46E  00FF 5204 4200 8020  •4•a•••n••R•B••
```

MacsBug would then set the dot address to the address starting after the first occurrence of 1234. If you then pressed Return (to repeat the command), MacsBug would output

```
Searching for 1234 from 000044BB to 001044BA
  0000B076  1234 4E75 4EB9 4081  0DA4 302E 000E 6758  •4NuN•@•••0.••gX
```

You will notice that in MacsBug 6.1, the end-of-range address does not remain constant but is equal to the sum of the new starting address plus the value you specified for *nbytes*.

In MacsBug 6.2, the behavior of the F command, which also applies to the FB, FW, FL, and FP commands, has changed. If you enter the same command (`F 0 100000 1234`), MacsBug 6.2 displays the following information:

```
Searching for 1234 from 000044BB to 000FFFFF
  0000B076  1234 4E75 4EB9 4081  0DA4 302E 000E 6758  •4NuN•@•••0.••gX
```

If you press Return a few more times, MacsBug reports the following:

```
Searching for 1234 from 000044BB to 000FFFFF
  0000B076  1234 4E75 4EB9 4081  0DA4 302E 000E 6758  •4NuN•@•••0.••gX
Searching for 1234 from 0000B078 to 000FFFFF
  0001D028  1234 702C 3F01 486E  FF5C 486E FEC6 487A  •4p,?•Hn•\Hn••Hz
Searching for 1234 from 0001D02A to 000FFFFF
  0001D0BA  1234 702C 3D41 FEA8  3F01 486E FF8E 486E  •4p,=A••?•Hn••Hn
```

You will notice that the end-of-range address remains constant. MacsBug 6.2 is behaving more reasonably in that it takes seriously your initial specification of a limited range. Of course, the end-of-range address is not underlined in MacsBug output; it is underlined in the examples presented above for your reading comfort.

The Find Byte, Find Word, and Find Long Commands

Another problem with the F command in MacsBug 6.1 is that it assumed that the size of the *expr* to look for was the smallest unit (byte, word, or long word) that would contain its value. For example, if you specified 00001234 for *expr*, the F command ignored the leading zeros and looked for the value 1234.

You can now use three additional commands, FB, FW, and FL , to prevent MacsBug from making what could be a bothersome assumption. These commands allow you to specify the exact width of the pattern that MacsBug looks for.

The syntax of the command remains the same except that you optionally add the letters B, W, or L to the F command to indicate the actual size of *expr*:

F[B|W|L] *addr nbytes expr*

B specifies that MacsBug should search for the byte value specified by *expr*.

W specifies that MacsBug should search for the word value specified by *expr*.

L specifies that MacsBug should search for the long word value specified by *expr*.

For example, if you enter,

```
FB 0 400 8
```

MacsBug 6.2 returns

```
Searching for 08 from 00000000 to 000003FF
 00000027  0840 8064 BA40 8021  0C40 8021 0E40 8021  •@•d•@•!•@•!•@•!
```

If you enter,

```
FW 0 400 8
```

MacsBug 6.2 returns

```
Searching for 0008 from 00000000 to 000003FF
 000003BA  0008 FFDF 0008 0000  0008 003C 2A1E 0000  ••••••••••••<*•••
```

If you enter

```
FL 0 400 8
```

MacsBug 6.2 returns

```
Searching for 00000008 from 00000000 to 000003FF
 000003B8  0000 0008 FFDF 0008  0000 0008 003C 2A1E  ••••••••••••<*••
```

The Find Pointer Command

The FP (Find Pointer) command allows you to find every occurrence of a pointer in the range of memory you specify. The syntax of the FP command is

FP *addr nbytes expr*

P specifies that MacsBug should search for the lower three bytes of *expr.*

A specific Find command for pointers is useful because in software releases prior to system software version 7.0, applications sometimes used the high byte of the long word containing an address to pass data. For example, the Memory Manager used the high byte of the long word containing the address of a relocatable block to specify whether the block was purgeable, locked, or a resource. This means that you cannot use the FL command to find every reference to an address because sometimes that address might be stored in its stripped form, 00xx xxxx, and other times in its non-stripped form, 02xx xxxx, 04xx xxxx, 0Exx xxxx, and so on. The following example shows how you would use the FP command to find all handles to a heap block and how you would check that the addresses returned were valid handle addresses.

Let's suppose that you want to find every reference to a heap block. For the sake of this example we will take the Geneva FOND resource, which is stored as a relocatable block in the system heap. Figure 3 shows the master pointer and some of the handles to this block.

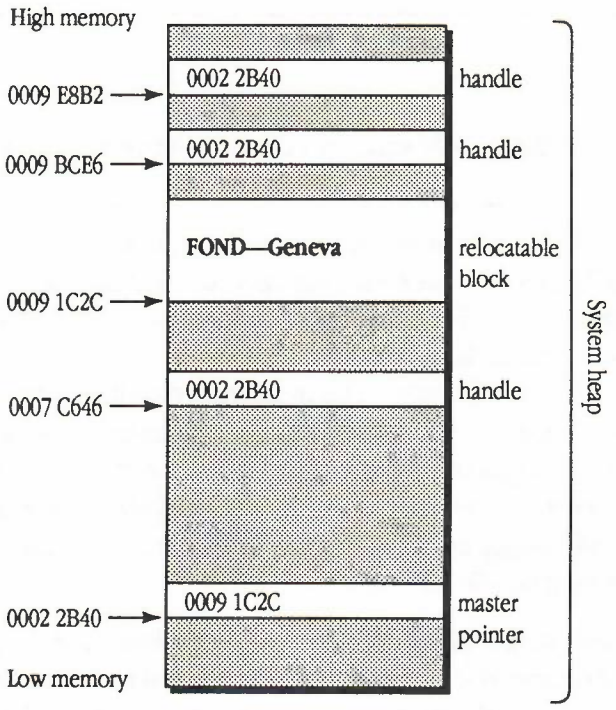
To obtain information about a block in memory you would use the HD command. However, since FOND resources are shared by all applications (and therefore stored in the system heap) you would first use the HX command to select the system heap and then enter

```
hd FOND
```

to obtain a display of all the FOND resources in the system heap. The following listing shows a partial output listing for the command. The information about the Geneva FOND resource is underlined.

```
Displaying the System heap
   Start     Length      Tag  Mstr Ptr Lock Prg  Type   ID   File      Name
00090BDC 00001046+02   R    00022B28       P   FOND   0014 0002      Times
00091C2C 00000060+00   R    00022B40       P   FOND   0003 0002      Geneva
000A3704 000002F0+00   R    00034B70       P   FOND   000D 0002      Zapf
```

■ **Figure 3** Pointers and Handles to a FOND resource



Notice that the address of relocatable blocks as well as the address of master pointers to the blocks are displayed in their stripped forms by the HD command. You can now use the FP command to find every handle to the block. For example, if you enter the command

```
FP 0 BufPtr^  00022B40
```

and press Return to repeat the command until MacsBug finds no further occurrences of the pointer, MacsBug displays every address where the pattern xx022B40 is stored.

```
Searching for xx022B40 from 00000000 to 004CDE71
 0007C646  0002 2B40 0000 0100  0100 0100 0100 0009  ••+@••••••••••••
Searching for xx022B40 from 0007C64A to 004CDE71
 0009BCE6  0002 2B40 0015 00B9  E004 1EE6 0002 2B34  ••+@••••••••••+4
Searching for xx022B40 from 000EAC5A to 004CDE71
 0028F464  7002 2B40 F606 7A00  7C00 206D FECC 2050  p•+@••z•|• m•• P
Searching for xx022B40 from 0028F468 to 004CDE71
 00377F12  7002 2B40 E93C 6000  0112 7002 B02B 0084  p•+@•<`•••p••+••
Searching for xx022B40 from 00377F16 to 004CDE71
 Not found
```

Note that MacsBug only looks for the lower three bytes of the pointer and assumes that 00022B40 and 70022B40 refer to the same master pointer. If you need to track down every handle to the block, you should make the same assumption because some routines strip the address and some do not.

To check whether an address returned by the FP command does in fact contain a handle to the block, you can use the DM command to display memory at that address. For example, if you enter

```
DM 0007C646 handle
```

MacsBug outputs the following information:

```
Displaying handle
 0007C646  00022B40 -> 60091C2C
```

That is, the value 00022B40 is stored at 0007C646. The value 60091C2C is stored at address 00022B40. 60091C2C looks suspiciously like the address of the Geneva FOND resource with a high byte of 60, which is what the Memory Manager would tack on to indicate an unlocked purgeable resource. Looking back at the output of the HD command we find that the Geneva FOND resource is indeed unlocked and purgeable.

Macros for the Find Command

Among the standard macros shipped with MacsBug 6.2 are the macros described in Table 1. You can use these macros with the Find command to specify common address ranges.

Table 1 Macros for the Find Command

Macro	Description	Macro Expansion
RamF RamFW RamFL RamFP	Defines RAM as the address range of the Find command. Example: `RamF 'Main'`	F 0 BufPtr^ FW 0 BufPtr^ FL 0 BufPtr^ FP 0 BufPtr^
SysF SysFW SysFL SysFP	Defines the System zone as the address range of the Find command. Example: `RamFW 1234`	F SysZone^ (SysZone^^-SysZone^) FW SysZone^ (SysZone^^-SysZone^) FL SysZone^ (SysZone^^-SysZone^) FP SysZone^ (SysZone^^-SysZone^)
ApF ApFW ApFL ApFP	Defines the Application zone as the address range of the Find command. Example: ApFP 0032e232	F ApplZone^ (ApplZone^^-ApplZone^) FW ApplZone^ (ApplZone^^-ApplZone^) FL ApplZone^ (ApplZone^^-ApplZone^) FP ApplZone^ (ApplZone^^-ApplZone^)
ZF ZFW ZFL ZFP	Defines the zone selected by the last HX command as the address range of the Find command. Example: ZFL 000A232B0	F TargetZone (TargetZone^-TargetZone) FW TargetZone (TargetZone^-TargetZone) FL TargetZone (TargetZone^-TargetZone) FP TargetZone (TargetZone^-TargetZone)

- The new variable TargetZone used in the Z Find commands described above is defined as the zone currently selected by the HX command. You can use it with other MacsBug commands to indicate a range.

New Heap Dump Display

The output for the HD command has been modified. If you enter HD, MacsBug 6.2 displays information in the following format:

```
Displaying the Application heap
    Start     Length       Tag  Mstr Ptr Lock Prg  Type   ID   File      Name
  • 0031CFE8 00000100+00   N
  • 0031D0F0 00000018+00   R   0031D0E4   L
  • 0031D110 00000032+02   N
  • 0031D14C 0001CD36+02   N
  • 00339E8C 0000BDFC+00   N
  • 0035232C 00000A60+00   N
  • 00352D94 00029800+00   R   0031D070   L
  • 0037C59C 00000006+02   N
    0037C5AC 0000009C+00   F
    0037C650 0000000C+00   F
    0037C664 0000000C+00   F
    0037C678 00000010+00   F
  • 0037C690 00000100+08   N
```

The new output provides additional information for a block's Length field by showing the length of the block in terms of two distinct operands; for example

```
00000100+08
```

The first number gives the actual length of the block (the size you asked for when you allocated the block); the second number indicates the number of bytes that have been added by the Memory Manager to meet the two requirements that

- A block must end at an even address.
- A block must have a minimum length of 12 bytes. (This requirement is different for 32-bit heaps.)

Previous versions of MacsBug displayed the length of a block as the sum of these two numbers. This display provides more exact information.

Summary of Changes to MacsBug Commands

Some MacsBug commands have been changed in slight ways. Table 2 describes these changes.

■ **Table 2** Changes to MacsBug Commands

Name	Syntax/Alternate	Description/Changes
ATP	ATP	If ATR is off, the ATP command plays back information from your most recent ATR
GT	GT *addr* [';*cmds*']	Go till *addr* is reached and optionally execute one or more commands. The *cmds* parameter is an addition of MacsBug 6.2.
Command-:	Command-D	Alternate way of entering command for use on German and Scandinavian keyboards. Command-: or Command-D displays a menu of procedure names.

Specifying Commands Using DebugStr

MacsBug defines a DebugStr trap that allows you to invoke MacsBug from your source program and to specify a message you want MacsBug to display when MacsBug is invoked.

Beginning with MacsBug 6.1, the DebugStr trap was extended so that you could include one or more commands in the message string to DebugStr. After MacsBug was invoked by the DebugStr trap, in addition to displaying a message specified with DebugStr, MacsBug could also execute the command(s) you included in the message for DebugStr. The syntax for the call is as follows:

DebugStr ("*string* [; *cmd*]...")

string is the message you want displayed.

cmd is a MacsBug command or macro.

Example: This routine would invoke MacsBug, display the message "Checking the Heap," do a heap check, and then resume execution of your program.

```
DebugStr ("Checking the heap; hc; g")
```

This next routine would display the message "Checking for segment loading," output information about the specified code segment, and then resume execution of your program.

```
DebugStr ("Checking for segment loading; hd code3; g")
```

Working with the Debugger Preferences File

This section describes the Debugger Preferences file which is included on the MacsBug 6.2 release disk. When you open the Debugger Preferences file, ResEdit (1.2 or later) displays a list of resources similar to that shown in Figure 4. Note that the display includes information about the number of resources of a certain type (count) as well as the total size of the resources of that type.

Note that *all* MacsBug resources have been placed in the Debugger Preferences file including the resource file used to provide help information.

■ **Figure 4** Debugger Preferences

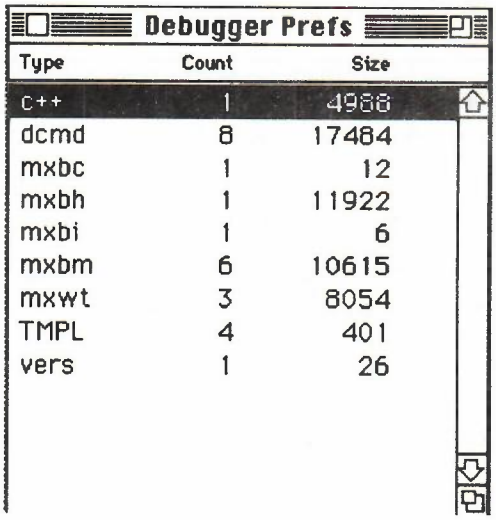


Table 3 describes the contents of the Debugger Preferences File.

■ **Table 3** Resources in the Debugger Preferences File

Resource	Contents
mxbh	MacsBug help messages.
mxbi	Specifies the size of the history buffer, the number of traps recorded by MacsBug, and the number of lines displayed in the PC area of the display.
mxbc	Specifies color display preferences.
mxbm	Defines the macros you can use to reference low memory globals and macros that define useful command lines.
mxwt	Defines the templates you use to obtain a more readable memory display
dcmd	Defines the dcmds you can use in addition to MacsBug commands.
C++	Used to unmangle C++ compiled files so that MacsBug can understand them.

Standard dcmds

Table 4 lists the dcmds included in the DebuggerPrefs file dcmd resource.

■ **Table 4** Standard dcmds

dcmd	Description
drvr [*refnum* | *num*]	Lists all the currently installed drivers or lists information for the specified driver.
file [*fRefNum* | "*filename*"]	Lists all open files or information about the specified file.
vol [*vRefNum* | *drvNum* | "*volumeName*"]	Lists all the volumes on line or displays volume information for the specified volume
vbl	Lists all the VBL tasks currently installed.
printf "*format*" *arg*...	Displays the arguments according to the format.

The printf dcmd is a formatted output command that behaves very much like the C programming language printf command. This section describes how you use the printf command for those of you who are unfamiliar with C and gives several examples of how you can use this command in debugging. To avoid confusion, please interpret any reference to the printf command in this section as a reference to the MacsBug printf dcmd, *not* the C printf command. The syntax of the printf command is

printf "*string*" *arg* [*arg*] ...

string is a combination of literals and conversion specification(s).

arg is an expression that is evaluated and converted according to the conversion specification to which it corresponds

The sample printf command below,

```
printf "Data for %d will be available on %s" 1990, Wednesday
```

produces the following output:

```
Data for 1990 will be available on Wednesday
```

A conversion specification consists of the percent symbol (%), which introduces the specification; an optional digit specifying the field width of the converted argument; and a conversion character specifying how the argument is to be represented.

Table 5 shows the meaning of the conversion characters you can use with the printf command.

▪Table 5 Conversion Characters for the printf dcmd

Conversion Character	Meaning	Example
d	decimal integer	93
o	octal integer	77
x	hexadecimal integer	2F
u	unsigned decimal integer	99
c	single character	q
s	string	application

If you're logging the output of a MacsBug session, you can use the printf command to make MacsBug output more intelligible. The following printf command,

```
printf "this application is %s" curapname
```

outputs

```
this application is Finder
```

This printf command

```
printf "this application's name contains %d characters" curapname^.b
```

outputs

```
this application's name contains 6 characters
```

If you are using the Debugger inline call to invoke MacsBug from within your source program, you can use the printf command with the DebugStr call to have MacsBug output key values during program execution.

This printf command

```
printf "Register A7 (%8x) points to word %x (= #%d)." RA7 RA7^.W RA7^.W
```

outputs

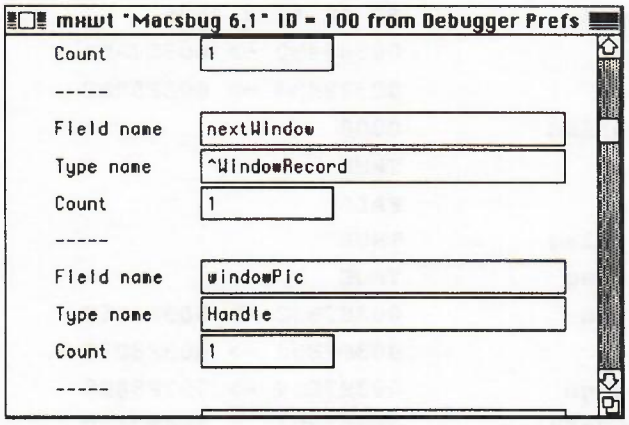
```
Register A7 (  4b7026) points to word 4080 (= #16512).
```

Note the use of 8 in the first specification (%8x) to specify the field width of the converted argument.

Constructing Linked Lists Using the mxwt Resource

MacsBug can display linked lists if you have marked the fields appropriately in your mxwt resource. Figure 5 shows the entry for the nextWindow field for the WindowRecord template. Note that the field type defines this field as a pointer to another window record.

■ **Figure 5** Linked list field entry in mxwt resource



If you use the dm (display memory) command from MacsBug to show you a window record; for example:

```
dm @WindowList WindowRecord
```

and the application you're debugging has several windows open, MacsBug outputs information similar to the following:

```
Displaying WindowRecord at 003E94CC
  003E94DC  portRect            #0 #0 #429 #516
  003E94E4  visRgn              003E7D0C -> 003E9570
  003E94E8  clipRgn             003E7D08 -> 003E9584
  003E9538  windowKind          0008
  003E953A  visible             TRUE
  003E953B  hilited             FALSE
  003E953C  goAwayFlag          TRUE
  003E953D  spareFlag           TRUE
  003E953E  strucRgn            003E7CB4 -> 003FF734
  003E9542  contRgn             003E7CB0 -> 003BCFA0
  003E9546  updateRgn           003E7CAC -> 003EC5E4
  003E954A  windowDefProc       080020D4 -> 20832A5C
  003E954E  dataHandle          003E7CA0 -> 003EC630
  003E9552  titleHandle         003E7CA8 -> 003FF24C  Untitled1
  003E9556  titleWidth          003C
  003E9558  controlList         003E7C94 -> 003FD6DC
  003E955C  nextWindow          003E7A50
  003E9560  windowPic           NIL
  003E9564  refCon              00000004
```

If you press Return, MacsBug displays information about the next window in WindowList, whose address (003E7A50) is given in the nextWindow field.

```
Displaying WindowRecord at 003E7A50
  003E7A60  portRect            #0 #0 #429 #506
  003E7A68  visRgn              003884B0 -> 003E7AF4
  003E7A6C  clipRgn             003884B4 -> 003F8DEC
  003E7ABC  windowKind          0008
  003E7ABE  visible             TRUE
  003E7ABF  hilited             FALSE
  003E7AC0  goAwayFlag          TRUE
  003E7AC1  spareFlag           TRUE
  003E7AC2  strucRgn            003E7B5C -> 003F8D68
  003E7AC6  contRgn             003E7B58 -> 003F8D7C
  003E7ACA  updateRgn           003E7B54 -> 003F88E8
  003E7ACE  windowDefProc       080020D4 -> 20832A5C
  003E7AD2  dataHandle          003E7B48 -> 003F8938
  003E7AD6  titleHandle         003E7B50 -> 003FD814 -> mbreleasenotes
  003E7ADA  titleWidth          006A
  003E7ADC  controlList         003E7B3C -> 003F8B5C
  003E7AE0  nextWindow          NIL
  003E7AE4  windowPic           NIL
  003E7AE8  refCon              00000003
```

(Portions of the output above are underlined for you reading comfort.) When the nextWindow field has a value of NIL and you ask MacsBug to show you the next window record, it displays the message "End of linked list."

Note also that MacsBug 6.2 displays complete information for fields defined as handles, for example the strucRgn and updateRgn fields shown in the previous listings.

You use the mxwt resource to define templates for structures that are unique to your application. If your structures include fields that are handles or pointers that you use to build linked lists, declaring them as such will allow MacsBug to display more detailed information about these structures, which should make debugging a whole lot easier.

Removing Resources to Gain Memory

If you *must* save space in memory, you can do so by deleting, moving or altering the following resources from the Debugger Preferences file. You should not remove the TMPL resource because this is the resource that ResEdit uses to display the other MacsBug resources. Without it you'll be eating hex dumps for breakfast, lunch, and dinner.

The effect of moving or altering the resources is described in Table 6:

- **Table 6** Effect of removing resources from Debugger Preferences file

Size (K)	Resource	Effect
5	C++	Needed only by C++ programmers for unmangling CFront generated code.
12	mxbh	You must rely on the documentation for help.
–	mxbi	MacsBug uses default values. If you reduce the size of the history buffer, you gain some memory but can save less information.
–	mxbc	Uses default black and white display.
8	mxbm	If you remove resources 101 and 102, you will no longer be able to reference low memory globals by name. Please examine the contents of the other mxbm resources before you remove or delete them.
8	mxwt	You will lose the use of templates to make sense of memory display.
1.6	dcmd	You will no longer be able to use these commands. You can remove dcmds individually. See Table 4 .

Using MacsBug Under A/UX

A/UX supports MacsBug 6.2. To install MacsBug, drag the MacsBug document and the DebuggerPrefs file into the system folder. When you are done, you should have the following two files:

```
/mac/sys/System Folder/MacsBug

/mac/sys/System Folder/DebuggerPrefs
```

Then reboot your machine.

To invoke MacsBug, you must press the `control-command-i` keys simultaneously. This will bring up the MacsBug display, and you can start working with MacsBug.

The only limitation you will encounter in working with MacsBug under A/UX is that you will not be able to use the Log command to log output to an Imagewriter. However, you can still log to a file.

As the version of MacsBug that runs under A/UX behaves identically to that running under the Macintosh system software, you can use the same documentation for your debugging: *MacsBug Version 6.1 Reference Manual* and this software release note.

▲ **Warning** Do not press the programmer's switch to invoke MacsBug . Under A/UX the programmer's switch is used to break into the A/UX kernel debugger; it will not get you into MacsBug. ▲