Cool MacsBug Tricks (an informal guide) By Macneil Shonle

- -
- - - - - - - - - - -

This guide is to help you in learning to use MacsBug. MacsBug is a system
extension that can help you debug your programs, it is free and is available
from Apple Computer. Note that the name MacsBug is an acronym for Motorola
advanced computer systems debuger.

This guide is a list of "tricks," but it is just the tip of the iceberg of
the cool stuff you can do. The guide will start off with some easy topics
aimed towards beginners and then it will go into some more advanced topics.
All of them are cool. Hopefully after reading this the help part of MacsBug
won't be so intimidating.

Number Conversion
MacsBug can be used as a quick hexadecimal to decimal converter, and vise
versa. I used to use a calculator DA, but now I just simply drop into
MacsBug and type in the number I want to convert and hit return.

Example: You want to find out what 0x3E is in decimal. When in MacsBug, type
in $3E and hit return. This will be the output:

$3E = $0000003E #62 #62 '***>'

The first number ($0000003E) is the value in hexadecimal that you just typed
in. The second number is what the value is as an unsigned decimal, the third
is the signed version. The set of characters in single quotes ('***>') is
the ASCII representation of the number, the null character is denoted with
the bullet.

You can find out the decimal/hexadecimal equivalent of any ASCII

character
by typing the letter balanced between two single quotes.

Example: Type in: 'A' and hit return. You will get #65 as your
answer.

By the way-The dollar sign means that the number is in hexadecimal. Much
like C's 0xXX notation, $XX is how hexadecimal numbers are represented in
assembly. Numbers in MacsBug will default to hexadecimal, except for when
the hexadecimal number is a command or a regster. For example: "ea" is the
command to restart the current application, when you type in ea it will try
this command, you must type in $ea in order to avoid this conflict.

Similarly, you have to type a # in order to express decimal numbers. You can
use the conversion method just described (type in the number, hit return) to
find out a decimal numberÕs corresponding hexadecimal number and
ASCII
character.

What Was My Monitor Size? HereÕs an impresive way to show a friend how many
pixels horizontally and vertically they have on their monitor (other than
looking at the manual, or something silly like that). Drop into MacsBug and
type in: dm @@MainDevice GDevice. This will show you the struct members of
the MainDevice (which happens to be a GDevice), you should see the gdPMap
indented, three lines below it will be bounds with four numbers to the right
of it. These four numbers are the top, left, bottom and right coordinates of
the monitor, respectively.

The dm command is short for display memory, after you type in dm type in the
address of the memory you want to display. MainDevice is a system global

that is a handle (a pointer to a pointer) to the main graphics device (the
one with the menu bar). The two @@ symbols are how you express double-indirection in MacsBug, in C you use "*" to express indirection (i.e.
de-referencing) which is in put in prefix notation. People who program in
Pascal can use the postfix indirection notation by saying "dm MainDevice^^
GDevice".

After you give the dm command the address, you give it the format you want
to see it diplayed as. You can use any number for the number of bytes you
want displayed, or you can say "Rect", for instance, to see the first eight
bytes of the memory in the form of a rectangle. You can also use: Byte,
Word, Long, SignedByte, SignedWord, SignedLong, UnsignedByte, UnsignedWord,
UnsignedLong, PString, CString, and PixMap, GDevice, RGBColor, CGrafPort and
any number of other templates you may have installed.

Example: if you know a rectangle is at address $00058EA6 and you
want to see

what its value is, all you have to do is type in "dm $00058EA6 Rect".

By the wayÑA template a layout of memory that MacsBug knows about (such as a
C struct or a Pascal record), you can type "tmp" to find out all of the
templates your version of MacsBug has.

Don't you hate it when you are working in an application, minding your own

business, when all-of-a-sudden the program quits and the system tells you an
error of type X occured? There are many applications made where you can look
up these numbers and find out what went wrong. MacsBug can also do this, all
you have to do is type error and then the error number. Keep in mind that
the error numbers the system gives you are decimal (not

hexadecimal), so you
should put a "#" in front of them.

Example: The sytem tells you: "An error of type 4 has occured," drop into
MacsBug and type "errorÊ#4", MacsBug will then output "$0004Ê#4ÊzeroÊdivideÊerror".

Note: This error feature is not in earlier versions of MacsBug, so you may
not have it.

The Simple Calculator You can use MacsBug as a simple calucator. LetÕs say
you need to know what seven times seventeen is, type in "#7Ê*Ê#17", and hit
return. The number 119 should now be on your screen. It will be hidden in
the line:

#7 * #17 = $00000077 #119 #119 '¥¥¥w'

The lower case letter w is the 119th ASCII character, as the previous line
shows us. LetÕs try another example, how about five plus six? You would type
in "#5 + #6", and hit return. You should then see:

#5 + #6 = $0000000B #11 #11 '¥¥¥¥'

MacsBug can also handle multiple operations at a time, like five plus six
plus ten. If you want to say something like five plus six times four
remember to put parentheses around the apropiate numbers. MacsBug has no

concept of orders of operations and itÕs quite possible for it to add before

it multiplies. So say this: "#5Ê+ (#6Ê*Ê#4)", which equals #29,
instead of

"#5Ê+Ê#6Ê*Ê#4", which equals #44.

You can use +, -, *, /, MOD for arithmetic operations. You can use AND (or
&), OR (or |), NOT (or !), XOR for boolean operations. And you can use = (or
==), <> (or !=), <, >, <=, >= for equality operations.

If you type in "#5Ê+Ê#4Ê=Ê#9" MacsBug will give you a one, meaning that the
equality you just said was true. If you said "#5Ê+Ê#4Ê=Ê#10", Macsbug will
give you a zero, meaning that the equality five plus four equals ten is
false.

Moving the Cursor Here is a cool trick to move the cursor. It done by
setting memory, the mouse tracking variables specifically. But IÕd like to
talk about setting memory beforehand. There are four commands in MacsBug to
set memory: SB (set byte), SW (set word), SL (set long), and SM (set
memory). You give each of these commands an address first, and then the
values of what you want to set the memory to. Example: There is a byte that
you have the address of that you want to set to ten, you should type in:

SB $XXXXXXXX #10

where $XXXXXXXX is the address of the byte. Another example: There is a long
that you have the address of that you want to set to "$4D616320", you should
type in:

SL $XXXXXXXX $4D616320

again, where $XXXXXXXX is the address of the long. You can use the SM
command the same way in the case that the length you want to set is not 1, 2
or 4 bytes long. You can use SW when you want to set a word (2 bytes).

If you are familiar with Points (the vertical and horizontal coordinates of
a point on the graf plane), you should know that they take up four bytes in
memory. The high two bytes (the high word) is the vertical coordinate, and

the low two bytes (the low word) is the horizontal coordinate. There are two
global variables that are both Points, one called MTemp, the other called
RawMouse, these variables are the information the Macintosh uses for
controling the cursor. You can set these points by using SL.

There is also a byte called CrsrNew, set this byte to 1 when you want to

notify the Macintosh that the cursor posistions have changed. This is how

you move the mouse to point (5,Ê6), near the upper-left corner of
the
screen:

SL MTemp $00060005 SL RawMouse $00060005 SB CrsrNew #1

Make sure MTemp and RawMouse have the same value. Now type Command-G to see
your newly moved cursor.

Recovering from a Hung Serial Port
Sometimes when you're AppleTalking or modeming and something goes wrong

(like you switch the modem off while data is being sent to it), the comptuer
will hang. The mouse will still move, but clicking will have no effect.
Here's the solution:

Drop into MacsBug. You should see the routine name "_vSyncWait" plus
something as the current location. If you don't, you probably hit the system
while it was doing something else. Hit Command-G to get back out of MacsBug,
and try again. After a few tries you should find _vSyncWait.

_vSyncWait is the routine that the system uses to wait for some input from a
serial port. If you can read assembly code, youÕll see that itÕs pretty
simple. HereÕs the dump of the significant part:

+0000 4080BB8C MOVE.W $0010(A0),D0 |
3028 0010

+0004 4080BB90 BGT.S _vSyncWait ; 4080BB8C |
6EFA

Register A0 is pointing to a system data structure, in which a word will be
cleared when the awaited input arrives. The MOVE.W instruction grabs this
word and puts it into register D0. The BGT.S instruction then Branches back
to the MOVE.W if the byte it just fetched is Greater Than zero. So it
happens that this byte is never going to arrive for whatever reason, but the
computer is going to wait for eternity. The secret to fixing this is to use
Command-T to go step along until the MOVE.W instruction is displayed as the
current instruction. Now use the sw command to set "@(A0+10)" to

zero:

sw @(A0+10) 0

Hit Command-T twice more. The MOVE.W instruction will take the zero you just
set into memory and put it in D0, so the D0 display on the left of the
screen should have its right four digits all zeros. Then when you execute

the BGT.S instruction, it should not go back to the MOVE.W since zero is not
greater than zero.

Hit Command-G to go. If this was the only byte the software was waiting for,
then it should continue running, although it may go a little crazy because
it's been suddenly disconnected from whatever peripheral it was talking to.
Quit the program, fix your hardware, and try again.

Shameless Strobe Light Trick
Okay, this is a really useless trick, but itÕs cool for at least a little
while. Go into MacsBug. If you have a single screen type in "swap", the
console should then say "Display will be swapped after each trace or step",

if it doesnÕt type in "swap" again. Swapping is when the screen switches
from the MacsBug console to the normal Macintosh screen. We want it to swap
after each trace or step, which is what we just did up above. Now we need it
to step, thereby swapping the screen, the "s" command (the step command) is
just what we need to do this. We want this to happend more than once, so we

type in: "s 100", which steps 100 times. Enjoy the show.

Warning: Swapping with a number like 1000 can render some machines, like my
PowerBook 165c, useless until it is all over with, so keep the numbers low
or the patience high.

GetKeys from within MacsBug
There is a routine in the Macintosh toolbox called GetKeys, this routine is
great for game programmers who want a reasonably fast way to read the
keyboard, without using (slower) events. The problem for C and C+
+

programmers using this routine is that the KeyMap type is a Pascal packed
array. Each bit of the packed array is designated to a certain key, the bit
is set to 1 if the key is down, and set to zero if the key is up. This array
takes up 16 bytes (128 bits). C cannot access the elements of the packed
array like a normal array, so the programmer has to mask out some bits to

get the result that he/she wants. There is a desk accessory named "GetKeys,"
that is made just for this case. The problem is, you might not be
on a

machine with that program on it.

Good thing MacsBug is able to help us. Here is how you locate the bit for
the letter "M": go into MacsBug and type in "dm KeyMap", but don't hit
return just yet. Now strike the escape key, this should swap the
screen.

Press and hold down the letter "M" on your keyboard, this should swap the
screen back. Now, while still holding down "M", press return. This is what
you should see:

Displaying memory from 0174
00000174 0000 0000 0040 0000 0000 0000 0000 0000¥¥¥¥¥@¥¥¥¥¥¥¥¥¥¥

The number "00000174" is the address of the KeyMap global variable. The next
set of numbersÒ0000Ê0000Ó is the first element of the C version of the

array, in other words, itÕs: "keyMap[0]". The next set of numbers "0040Ê0000" is the second element of the array, keyMap[1]. The next group of
8 hexadecimal digits is the third element (keyMap[2]), and the last group of
8 hexadecimal digits is the fourth element (keyMap[3]). The series of
bullets is what the array looks like in ASCII form. In the second group

("0040Ê0000") there is a 4 in the midst of all of those zeros. This is the
bit that is set to 1 whenever the "M" key is held down. So, to see if the
"M" is down from within C we will do this:

KeyMap keyMap;
GetKeys(keyMap);

if(keyMap[1] & 0x00400000)
DoMKeyDown();

The Lost Paper
I was once typing in some text in a word processor, when the computer
suddenly crashed on me. I didn't save a copy on to the hard-disk yet. I had
to restart the computer and type it all over again. But wait, the paper is
still in the machine I thought to myself. You see, when you restart, all of
the computerÕs memory doesn't get cleared, it just stays to what it was
until it gets replaced with other information, much like the

behavior of a
hard-disk. I had one thing going for me, I had MacsBug installed. Here are

the steps I took to recover the paper:

First, I logged all of the work I was doing in MacsBug to a file. I did this
using the log command. All you need to give the log command is the name of
the new file to log to. I named it MyPaper. Good, now all of my MacsBug
session will be on the hard-disk so I can open it up with a normal text
editor when IÕm done.

Next, I needed to find where in memory my paper was. I did this using the
"f" command. The first two parameters for this command is the range in
memory in which you want MacsBug to search through. I wanted to search
through all of my memory, which is 8 megs on my machine, so I typed in:
"fÊ0Ê(400Ê*Ê400Ê*Ê8)Ê"any string". Where 0 is the beginning of memory and 8
megs is the top of it. (Note: "400Ê*Ê400" is exactly one megabyte
of
memory.) The last parameter is the search string, balanced between two
single quotes. I wanted to pick a distinct string, otherwise I would have
found other parts of memory, which would take longer to do. I knew the most
famous mammal, the aardvark, was mentioned in my paper, so I typed in this
for the find command:

f 0 (400 * 400 * 8) "aardvark"

MacsBug then started searching for me. It came up with a small memory dump
of something with the word arrdvark in it, but the words after it were not

mine, which meant that I found another part of memory instead of my paper. I
hit return to tell MacsBug to keep on searching. MacsBug then came up with a

dump from my paper:

Searching for "aardvark" from 00000000 to 007FFFFF
00358200 6161 7264 7661 726B 8000 0000 0000 002C aardvark answer, Which was
very good news indeed! This told me that the string "aardvark answer" could
be found at address 00358200. (I got this address from the leftmost number
given.)

Now that I knew where it was, the rest my task would be easy. I used

MacsBugÕs display ASCII command to show me what came after it, by typing in:
dma 00358200. You might not have this command, in which case youÕll have to
use plain old dm, instead of dma. I hit return until my paper was done being
displayed.

Note: You can subract a number from the address to see what was before it.

I then typed in "log" again, which closed my log. Finally, I went out of
MacsBug and opened the log file with SimpleText. Remember, the log had my
whole session not just the paper so I had to delete the addresses and such
from it, which really isn't that hard to do, if you know how to use your

mouse and your delete key efficiently. The paper was saved.

Warning: In your search you might stumble upon MacsBugÕs very own

memory,

with its own copy of your search string. To get out of this loop, start the
search over again with the base search address being outside of MacsBugÕ's
memory. Credits
The authors of this are Macneil Shonle and Dustin Mitchell of The Syzygy
Cult, a programming group that makes games and utilities. Dustin reviewed
this and submitted the Recovering from a Hung Serial Port section, thanks

Dustin!

Email MacneilS@aol.com if you have some input on how I can make Cool MacsBug
Tricks even better; nothing is too small to mention. Thanks for reading!

- -
- - - - - - - - - - -

Copyright© 1995, Macneil Shonle. All rights reserved.