NATIONAL OS9 NEWSLETTER

Editor : Gordon Bentzen
8 Odin Street,
SUNNYBANK Qld. 4109.
(07) 345-5141

NOVEMBER 1988

EDITOR : Gordon Bentzen
HELPERS : Bob Devries and Don Berrie

SUPPORT : Brisbane OS9 Level 2 Users Group

Another newsletter hits the streets!

This here is the fifth one, and we ain't about to stop here. In this issue you'll find some more of Don Berrie's Disk Zapper, and also the explanation of the workings of the C programme from last month. Also, it has been brought to my attention that there's some people in the national user group who are new to the world of OS9. Well, we'd like to provide something of interest for every-one, so there's an section in this newsletter for those people too.

We would like to see some input from other people in the group to add to our collection of material. One avenue open to us is a section dealing with problems of OS9 users around Australia. We would like to print these problems in the newsletter, along with whatever answers we can provide. So get those problems on paper, and send them to us. Your problem might be concerning other people, too, and the answers could help others.

I'm sure that information about running OS9 on other systems besides the Colour Computer would make interesting reading for us all. How about it, all you other system users?

Well here's the November issue. I hope you all learn something from it.

ZAP (Part 2)

```
**************************************************************************
                Disk Zapper Program for CoCo OS9 Level II
                    Copyright (c) 1988 by D.A. Berrie
                Released to the Public Domain - March 1988.
**************************************************************************
```

Program Description :

Zap is a sector based disk-zapper for use with CoCo OS9 Level II. It is presented as Basic09 code and consequently needs access to the Basic09 module to execute. The program also needs access to the Gfx2 module in order to do the windowing and other screen manipulations. To avoid conflicts with other windows which may already be active, the program uses the /W window descriptor. The program sets up a new window and then directs all output to it, thereby avoiding the necessity to run the process from a window of a particular type to give the correct display. This dictates that both the /W descriptor, and one other free window descriptor need to be available to the program.

The program has inbuilt help messages and is easy to operate. A word of CAUTION. You can make permanent changes to your disk structure, including the possibility of making a disk UNREADABLE. If you are unsure of what you are doing, do not use the "M" (for modify) option from the main menu-bar.

For a description of the floppy disk setup under OS9, read Chapter 5 of the Technical Reference Section of the Level II Manual.

System Requirements :

OS9 L2; 512K; grfint or windint module; basic09 (or runb for a packed version); gfx2; syscall; shell; display; xmode; and dir modules.

Instructions :

Type in the program after starting Basic09 with 30K of memory. You should end up with 18 separate procedures.

zap	helpmess	scn	winopen
winclose	ascii	change	swopen
swclose	helpmess2	modify	closerr
calc	header	getsec	getdev
directory	getdevnam		

The best way to run the program is to pack it into I-code and run it using RunB. After you have typed in the code, (and Saved it just in case) simply type pack# and the modules will be merged, and saved in your execution directory under the name zap.

Program Usage : zap <CR>

Disclaimer :

As the use of this program is beyond the influence of the author, no responsibility can be accepted. However, the program does work, if used correctly, and has been subjected to extensive error testing.

Help :

Should you experience any problems, or have any questions about the operation of the program, please feel free to contact me on (07) 375-3236.

Please feel free to distribute copies of this program. Cheers .... Don Berrie.

```
    PROCEDURE helpmess2
    ON ERROR GOTO 100
    DIM title:STRING[40]
    PARAM wpath:BYTE
    PARAM key:STRING[1]
    PRINT #wpath,"      ";
    title="MODIFY OPTION"
    RUN header(wpath,title)
    PRINT #wpath,
    PRINT #wpath,
    PRINT #wpath,"   USE :"
    PRINT #wpath,
    PRINT #wpath,"ARROW keys for"
    PRINT #wpath,"     cursor movement"
    PRINT #wpath," 2-digit Hex. number"
    PRINT #wpath,"     to change byte"
    PRINT #wpath,"<H> to display"
    PRINT #wpath,"     this menu"
    PRINT #wpath,"<W> to write changes"
    PRINT #wpath,"     to disk"
    PRINT #wpath,"<Q> to return to"
    PRINT #wpath,"     main menu"
    PRINT #wpath,
    PRINT #wpath,"Press Any Key to Continue"
    GET #wpath,key
    END
100 RUN closerr(wpath)
    END
```

```
PROCEDURE modify
ON ERROR GOTO 100
BASE 0
PARAM wpath:BYTE
PARAM secdat(256):BYTE
PARAM iblkno:INTEGER
PARAM PATH:BYTE
PARAM NAME:STRING[4]
PARAM flag:INTEGER
DIM x,y,x1,y1:INTEGER
DIM key:STRING[1]
DIM meter:REAL
x=10 \x1=60 \y=8 \y1=8
meter=iblkno
RUN winopen(wpath,x,y,x1,y1)
IF flag=0 THEN
  PRINT #wpath,"    >>> NO CHANGES MADE <<<"
  PRINT #wpath,
  PRINT #wpath,"   Press Any Key to Continue"
  ELSE
  PRINT #wpath,"WRITE CHANGES TO DISK"
  PRINT #wpath,
  PRINT #wpath,"       ARE YOU SURE   (Y/N) :";
  ENDIF
GET #wpath,key
IF key="Y" OR key="y" THEN
  OPEN #PATH,NAME
  meter=meter#256
  SEEK #PATH,meter
```

```
    PUT #PATH,secdat
    CLOSE #PATH
    ENDIF
  RUN winclose(wpath)
  RUN gfx2(wpath,"CURXY",5,22)
  PRINT #wpath,"insert 32 blanks  instead of this
    line";
  END
100 RUN closerr(wpath)
  END

  PROCEDURE closerr
  ON ERROR GOTO 100
  PARAM wpath:BYTE
 10 SHELL "display 1b 23"
  GOTO 10
100 END

  PROCEDURE calc
  BASE 0
  PARAM wpath:BYTE
  DIM title:STRING[40]
  DIM result,num1,num2:INTEGER
  DIM op1,op2:STRING[6]
  DIM keypress:STRING[1]
  RUN gfx2(wpath,"curoff")
  8 PRINT #wpath,CHR$($0C)
  PRINT #wpath,"      ";
  title="HEX CALCULATOR"
  RUN header(wpath,title)
  PRINT #wpath,
  PRINT #wpath,"USE : 0-9 A-F + - * / <cr>"
  op1="" \op2=""
  result=0
  PRINT #wpath,
  ON ERROR GOTO 10
  GOTO 11
 10 RUN gfx2(wpath,"curxy",6,5)
  PRINT #wpath,"*** ERROR"
  RUN gfx2(wpath,"bell")
 11 op1=""
  keypress=""
  REPEAT
    op1=op1+keypress
    RUN gfx2(wpath,"curxy",6,5)
    PRINT #wpath,"         "
    RUN gfx2(wpath,"curxy",15-LEN(op1),5)
    PRINT #wpath,op1
    RUN gfx2(wpath,"color",1)
    GET #wpath,keypress
    IF keypress="Q" OR keypress="q" THEN
      END
      ENDIF
    RUN gfx2(wpath,"color",0)
    UNTIL ASC(keypress)=$2A  OR ASC(keypress)=$2B
OR ASC(keypress)=$2D OR ASC(keypress)=$2F
  RUN gfx2(wpath,"curxy",15-LEN(op1),5)
  RUN gfx2(wpath,"color",6)
  PRINT #wpath,op1
  RUN gfx2(wpath,"color",0)
  op1="$"+op1
  num1=VAL(op1)
  RUN gfx2(wpath,"curxy",16,6)
  PRINT #wpath,keypress
  operator=SUBSTR(keypress,"+-/*")
  PRINT #wpath,
  ON ERROR GOTO 12
  GOTO 13
 12 RUN gfx2(wpath,"curxy",6,7)
  PRINT #wpath,"*** ERROR"
  RUN gfx2(wpath,"bell")
 13 op2=""
  keypress=""
  REPEAT
    op2=op2+keypress
    RUN gfx2(wpath,"curxy",6,7)
    PRINT #wpath,"         "
    RUN gfx2(wpath,"curxy",15-LEN(op2),7)
    PRINT #wpath,op2
    RUN gfx2(wpath,"color",1)
    GET #wpath,keypress
    IF keypress="Q" OR keypress="q" THEN
      END
      ENDIF
    RUN gfx2(wpath,"color",0)
    UNTIL ASC(keypress)=$0D
  RUN gfx2(wpath,"curxy",15-LEN(op2),7)
  RUN gfx2(wpath,"color",6)
  PRINT #wpath,op2
  RUN gfx2(wpath,"color",0)
  op2="$"+op2
  num2=VAL(op2)
  ON ERROR GOTO 100
  IF operator=1 THEN
    result=num1+num2
    ENDIF
  IF operator=2 THEN
    result=num1-num2
    ENDIF
  IF operator=3 THEN
    result=num1/num2
    ENDIF
  IF operator=4 THEN
    result=num1*num2
    ENDIF
  PRINT #wpath,
  PRINT #wpath," RESULT : ";
  RUN gfx2(wpath,"revon")
  RUN gfx2(wpath,"color",0,2)
  PRINT #wpath,"$";
  PRINT #wpath USING "h4",result
  RUN gfx2(wpath,"color",0,1)
  RUN gfx2(wpath,"revoff")
  PRINT #wpath,
```

```
    PRINT #wpath,"     Press Q to Quit"
    PRINT #wpath,"   Any Key to Continue"
    GET #wpath,keypress
    IF keypress="q" OR keypress="Q" THEN
      RUN gfx2(wpath,"curon")
      END
      ENDIF
    GOTO 8
    END
100 RUN gfx2(wpath,"curon")
    RUN closerr(wpath)
    END

    PROCEDURE header
    PARAM wpath:BYTE
    PARAM title:STRING[40]
    RUN gfx2(wpath,"color",2,3)
    PRINT #wpath,title
    RUN gfx2(wpath,"color",0,1)

    PROCEDURE getsec
    PARAM wpath:BYTE
    PARAM maxblock:INTEGER
    PARAM blkno:REAL
    DIM hblkno:STRING[25]
  1 PRINT #wpath,"SECTOR NUMBER (max:$";
    PRINT #wpath USING "H4>",maxblock;
    PRINT #wpath,") :";
    INPUT #wpath," ",hblkno
    hblkno="$"+hblkno
    ON ERROR GOTO 1
    blkno=VAL(hblkno)
    IF blkno>maxblock OR blkno<0 THEN
      GOTO 1
      ENDIF
    RUN winclose(wpath)
    END

    PROCEDURE getdev
    BASE 0
    ON ERROR GOTO 10
    PARAM wpath:BYTE
    PARAM name:STRING[4]
    PARAM path,secdat(256):BYTE
    PARAM maxblock,ident:INTEGER
  5 INPUT #wpath,"RBF Device Name : ",name
    IF LEFT$(name,1)<>"/" THEN
      name="/"+name
      ENDIF
    IF RIGHT$(name,1)<>"@" THEN
      name=name+"@"
      ENDIF
    OPEN #path,name
    SEEK #path,1
    GET #path,secdat
    maxblock=secdat(0)*256+secdat(1)-1
    ident=secdat(14)*256+secdat(15)
    END
 10 PRINT #wpath,"*** - DEVICE NAME REQUIRED"
    GOTO 5

    PROCEDURE directory
    PARAM wpath:BYTE
    DIM winnam:STRING[32]
    DIM title:STRING[40]
    DIM key:STRING[1]
    DIM pathlist:STRING[40]
 10 RUN gedevnam(wpath,winnam)
    pathlist=""
    ON ERROR GOTO 99
    PRINT #wpath,"   ";
    title="EXTENDED DIRECTORY"
    RUN header(wpath,title)
    PRINT #wpath,
    PRINT #wpath,
    INPUT #wpath,"Directory Pathlist : ",pathlist
    IF pathlist="" THEN
      END
      ENDIF
    IF LEFT$(pathlist,1)<>"/" THEN
      pathlist="/"+pathlist
      ENDIF
    SHELL "XMODE "+winnam+" PAU=20 PAUSE"
    SHELL "DIR "+pathlist+" E >"+winnam
    SHELL "XMODE "+winnam+" -PAUSE"
    PRINT #wpath,"Press Any Key to Continue"
    GET #wpath,key
    END
 99 PRINT #wpath,CHR$($0C)
    PRINT #wpath, \ PRINT #wpath, \ PRINT #wpath,
    PRINT #wpath,"               *** "; ERR; "
 ***"
    PRINT #wpath,
    PRINT #wpath,
    PRINT #wpath,"               CHECK SYNTAX"
    PRINT #wpath,
    PRINT #wpath,"          Press Any Key to
Continue"
    GET #wpath,key
    GOTO 10

    PROCEDURE gedevnam
    TYPE registers=cc,a,b,dp:BYTE; x,y,u:INTEGER
    DIM regs:registers
    PARAM wpath:BYTE
    PARAM winnam:STRING[32]
    DIM i:INTEGER
    DIM callcode:BYTE
    regs.a=wpath
    regs.b=$0E
    regs.x=ADDR(winnam)
    callcode=$8D
    RUN syscall(callcode,regs)
    FOR i=1 TO 32
```

```
    EXITIF MID$(winnam,i,1)>CHR$(128) THEN
      winnam="/" + LEFT$(winnam,i-1)  + CHR$
 (ASC(MID$( winnam,i,1))-128)
      ENDEXIT
    NEXT i
  END
```

```
*********************
*        ROGUE      *
*  SOFTWARE REVIEW  *
*********************
```

By Nickolas Marentes (CoCo3 Commercial Programmer)

Oh wow! Another new game for my CoCo3 and it's from Epyx, the mob who did "Koronis Rift" (which I reviewed last issue). This has gotta be good!

The packaging states "Rogue is so full of unpredictable monsters, ever-changing magic and hidden dangers that it's never the same game twice" and "You could spend hundreds of hours playing it...and you still wouldn't uncover all its secrets". Sounds impressive so without further ado, I opened the package and began to delve inside. Once inside I extract a manual and a disk. Opening the manual I...aaw what the heck! Rather than beat around the bush, I'll tell you right now that this program is absolute $%Z$$. Whatever you do DON'T buy this program. I don't think that this program is even worth "pirating"! (great way to cut down software piracy).

NEGATIVE POINTS:

There are three versions of this program on the disk. One uses the 40 column text screen, the other the 80 column text screen. Both look dull. I find it hard to imagine an "8" is the hero, ":" the floor, "/" a door, "%" as food and so on through 43 different symbols. The third version of the program had hope (HAD). I quote "MAKEGW - Opens a full screen graphics window that allows you to play Rogue with graphic images on screen for many of the items in the game. It looks good!!" unquote. Well, it was better, but still dull, especially in black and white (I thought I bought a COLOR computer?). By the way, you need 512K to use this option. Wasteful isn't it. The pictures on the back of the packaging look great compared to the actual screen. The program is very hard to control in that there are so many keys to control the "action". Sound...well, it sounds better with the volume turned down. Yes, that good! The price of this package is extraordinarily high for a program of its calibre. At $49.95, Tandy are really pushing their luck. I could go on but this review is only supposed to go for one page so I'll get on to the paragraph.

POSITIVE POINTS:

Um...well...er...the packaging looks all right !?

CLOSING COMMENTS:

I hope Tandy sack the guy who passed this program through the Computer Marketing division. More programs like this and a lot of CoCo3 owners are going to dump their CoCo for a Commodore or Atari. It's also bad exposure for the OS-9 operating system of which this program is running under. Luckily for the author, his name isn't printed anywhere. I can understand why this program is claimed to be "The college mainframe classic". In those days, computer games and graphics were still quite archaic. I suppose this program could be classified as "a blast from the past", it certainly is primitive enough. As one CoCo guru said to me after seeing it, "this one's a dud!". I agree.

STARTING OUT WITH OS9

Where Do I Start?

Lots of people must be asking themselves that question when they first look at OS9, whether it be Level One or Level Two. The answer, of course, is 'at the beginning'. Oh, sure, but just where is that? Well, when you purchase OS9, you get some disks, and a manual. You should at least read the first chapter of the manual first! This will tell you how to start OS9 running on your computer. If you are starting with Tandy's version of OS9, it is supplied on 35 track, single sided disks. This combination will work on all disk drive combinations unless you have 80 track drives that only work on 80 tracks and are not 'switchable'. However lacking in space the 630 sector disk is, it is a good starting point.

Get the system going. Of course, you should use a BACKUP of the original disks. You can create the backup on a colour computer by using the Disk Basic command 'BACKUP'. Of course this will only work on the disks as supplied by Tandy in the original format, not with previously modified system disks.

Try out some commands. The most obvious one is the 'DIR' command. It behaves mostly like the same command on most computers-it tells you what is on the disk. Well, not quite everything on the disk, but at least what is in your current 'DATA' directory. You see OS9 can have more than one of these directories, each with a different name, and they can all contain different types of files if you like. However, back to the one we're looking at. When you start up OS9 you will be looking at the 'ROOT DIRECTORY' of drive zero which is called '/D0'. That is the name of the drive and the name of the directory. In this directory you will have some files like 'startup' and 'OS9Boot' and some directory names like 'CMDS'.

How do you tell which is which you ask? Well the convention is to name directories in UPPERCASE and files in lowercase, but if you forget, you can get the 'DIR' command to tell you. Try typing 'DIR E'. What you get now is a bit more information about the files in the directory. Have a look at the one I have below.

```
 Directory of /d1   21:06:22

Owner Last modified Attributes Sector Bytecount Name
----- ------------- ---------- ------ --------- ----------
   0  87/02/16 1648  ----r-wr      A      69E3 OS9Boot
   0  87/02/16 1649  d-ewrewr     75       5C0 CMDS
   0  87/02/16 1654  d-ewrewr     7E       140 SYS
   0  86/08/13 1447  ----r-wr    237        C0 startup
   0  86/10/22 1606  ----r-wr    239       117 window.t38s
   0  86/10/22 1605  ----r-wr    23C       168 window.t80s
   0  86/10/22 1628  ----r-wr    23F       2B0 window.glr4
```

Here is what is displayed with the 'E' extension. First, the user number, in this case 0, which is you, the superuser, then the date the file was last changed, and then to the meat of our question, the attributes. There are eight attributes used. They are Directory, Non-sharable, Public execute, Public write, Public read, Owner Execute, Owner Write, and Owner read. The first attribute tells us whether the name in the directory is in fact a directory name. All the other attributes must also be set, except for Non-share so that we can use the sub-directory to store files in. The next bit of information is the sector number where the file or directory starts and the next the length of the file. All numbers are in HEXADECIMAL. The last is of course the name of the file.

Filenames can be up to 29 characters long with OS9, but remember, you have to type those names when using the Copy command etc. Filenames can have any characters except / # ! $ & * + . Filenames must start with a letter not a number. You can use the underscore and the period but no spaces. So the name Fruit_and_veges is valid but Fruit and veges is not.

Executable files (read programmes) should be stored in the CMDS directory, and OS9 looks for a directory called '/D0/CMDS' when it starts, so that it can find these files. You may change the 'execution directory' to some other directory once the system has started. Similarly, data files and 'shell scripts' can

be stored in any data directory. The file called startup must be in the root directory for OS9 to find it. What's in the startup file? Well, here's a listing of the one from the disk supplied with Level two OS9.

```
* Echo welcome message
echo * Welcome to OS-9 LEVEL 2 *
echo * on the Color Computer 3 *
* Lock shell and std utils into memory
link shell
* Start system time from keyboard
setime </1
date t
```

You'll notice this one has some comments (starting with an asterisk) in it so that we can see what it does. The 'echo' command (line 2) will print what is on the line after it to the screen. The link command makes sure you can't remove the shell programme (and its merged partners) from memory. Then you get to input the time and date, the '</1' means to get data from the keyboard. After you've input the date it is displayed back at you (just in case you got it wrong).

When OS9 Level two starts, it looks for two files, one is the shell script 'startup' and one is an executable file called 'autoex'. Level one just looks for 'startup'. The 'Autoex' file is useful. In fact no file called by that name exists on your system disks, but you can rename any executable file in the CMDS directory to 'autoex' and it will automatically load and execute that file.

Next time I'll discuss some more about this Operating System of ours and what you can do to customise it for your computer setup.

Bob Devries.

Multi-Vue and the C Programming language. (Part 2.)

Because of lack of space, I was not able to put comments into the source code for the mouse driven pull-down menu window programme. I will try to make up for that this month.

The first part of the source listing, starting on page 3 of the October issue, is a list of include files, which should be in the directory '/D0/DEFS'. The rest of the lines on that page are definitions, which are replaced in the source with their values (e.g. UPDATE is replaced by 3) by the 'C' pre-processor.

The left column on page 4 sets up a number of structures with pre-defined values. These, as you can see, are the names in the pull-downs. The one with 'Application' in it is seen only rarely. If you are quick of eye, you can catch it as you use the 'clear' key to switch to and from other windows.

The function at the top of the right column on page 4 is the interrupt handler. That is, it receives the values sent by OS9 when a keyboard interrupt (BREAK or CTRL BREAK) is sent, or when you push the mouse button. All it really does is store that value in a variable for us to use later.

Now we come to main(). We set up a few local variables, set the system to be unbuffered input and output (with the setbuf() function) and turn off the text cursor. CurOff is a function in the CGFX library which should be in the directory /D1/LIB.

Next we set up a framed window on the existing device window (by using stdout). If the current window is the wrong type and we cannot complete this part, we return an error and exit. Next we set up the interrupt handler routine. We select the graphics cursor next. The code words used for this are pre-defined in the header file called 'buffs.h'. A call to _ss_gip then sets up the type of mouse we want, and which joystick

port to look for it in. In this case I chose the right socket and used the hires adaptor. _ss_mous then tells the system how often to read the joystick and the timeout for the button. Now we tell the OS9 kernel what signal to send when the button is pressed. I chose 10, but any value above 3 will do. We then initialise the interrupt variable to 0, and go into the main loop of the programme.

We again use the msig function. Actually, the first one is probably superfluous. We check the sigcode variable. If it is 0 we go to sleep until another interrupt comes along. If it equals 10 (MOUSSIG) then we reset it to 0 and read the mouse packet using a call to the _gs_mous function, to see whether the pointer is on the control area of the window, that is, the bar at the top. If it is, then we use the switch and case construct of C with the value returned by _gs_msel to give us the selected pulldown number. The number returned by menu select is defined in the data structures on page 3, as is the number of the item of the pulldown. Notice here, that the part from the time that you press the button on the control bar till the time you push it again on a selected item of the pulldown, is all handled by the Windint I/O driver. Pushing the cursor arrow is also fully automatic because we chose to make it FOLLOW the mouse. All that this programme does now is to use the values returned by Windint to select the correct function in the rest of the programme. By the way, the left most square on the control bar is actually provided by windint and returns the value MN_CLOS, which provides a way to quit. It does not perform a pulldown. The programme now goes off to do the function which was selected from the pulldown menu. When it returns, it does it all again, until the variable 'quit' is TRUE (1). Then the graphics pointer is turned off, the signal handler is released, the text cursor is restored to normal, and normal window is selected, and the programme ends.

The rest of the functions are reasonably self-explanatory, and each function is called with the value of the pulldown item as its parameter. I hope this explanation fills the void I left last month. If any of you have problems or suggestions, feel free to ring me or drop me a line. My address is:-

21 Virgo Street,
Inala. Qld. 4077

and the phone number is (07) 3727816.

Regards, Bob Devries.

RGB Patch ! Puts OS9 into RGB mode when you boot up.

This patch will eliminate the need to call the 'montype' command from the startup file. It is especially useful for those who have done the modification to boot straight into the 80 column screen on startup. It will produce the correct colours immediately and not wait until the montype command is issued. There is one 'gotcha' with this patch, any future montype commands will not produce anything else other than RGB. If this presents no problems, patch away. You can use the following shell script to make life easier. But first, the following modules should be in memory or in the execution directory.

Echo
Load
ModPatch
Save
Attr
Unlink

```
-x
echo patching VDGInt
modpatch -s <VDGInt_Mod
echo saving new VDGInt
save VDGInt.io
echo changing attributes...
attr VDGInt.io e pe -a
echo patching GrfDrv
modpatch -s <GrfDrv_Mod
echo saving new GrfDrv
save GrfDrv
echo changing attributes
attr GrfDrv e pe -a
echo finished
```

That was the first file. You will notice that in line 2 and 8 the modpatch utility is called with input redirected from another file. These files are is as follows:-

```
# VDGInt_Mod #

1 VDGInt
c 0115 26 20
v

# GrfDrv_Mod #

1 GrfDrv
c 08D1 26 20
v
```

My thanks to Don Berrie for the code for this one,

Regards,
Bob Devries.