Unlink <modname> Usage : Unlinks module(s) from memory @WCREATE Syntax: Wcreate [opt] or /wX [-s=type] xpos ypos xsiz ysiz fcol bcol [bord] Usage : Initialize and create windows Opts : -? = display help -z = read command lines from stdin -s=type = set screen type for a window on a new screen @XMODE Syntax: Xmode <devname> [params] Usage : Displays or changes theparameters of an SCF type device @COCOPR Syntax: cocopr [<opts>] {<path> [<opts>]} Function: display file in specified format gets defaults from /dd/sys/env.file Options : -c set columns per page -f use form feed for trailer -h=num set number of lines after header -l=num set line length -m=num set left margin -n=num set starting line number and incr -o truncate lines longer than lnlen -p=num set number of lines per page -t=num number of lines in trailer -u do not use title

AUSTRALIAN OS9 NEWSLETTER

EDITOR	Gordon Bentzen	(07) 344-3881
SUB-EDITOR	Bob Devries	(07) 372-7816
TREASURER	Don Berrie	(07) 375-1284
LIBRARIAN	Jean-Pierre Jacquet	(07) 372-4675
	Fax Messages	(07) 372-8325
SUPPORT	Brisbane OS9 Users Group	

-u=title use specfied title -x=num set starting page number -z[=path] read file names from stdin or <path> if given @CONTROL Syntax: control [-e] Usage : Control Panel to set palettes, mouse and keyboard parameters and monitor type for Multi-Vue. Selectable from desk utilities menu as the Control Panel. Opts : -e = execute the environment file @GCLOCK Syntax: gclock Usage : Alarm clock utility for Multi-Vue. Selectable from desk utilities menu as Clock. @GCALC Syntax: gcalc Usage : Graphics calculator utility for Multi-Vue. Selectable form desk utilities menu as Calculator. @GCAL Syntax: gcal Usage : Calendar/Memo book utility for Multi-Vue. Selectable as Calendar from the desk utilities menu. @GPRINT Syntax: gprint Usage : Printer setup utility for Multi-Vue. Lets user graphically

ADDRESSES

Editorial Material:
Gordon Bentzen
8 Odin Street
SUNNYBANK Qld 4109

Library Requests:
Jean-Pierre Jacquet
27 Hampton Street
DURACK Qld 4077

CONTENTS

Volume 7 **May 1993** **Number 4**

AUSTRALIAN OS9 NEWSLETTER
Newsletter of the National OS9 User Group
Volume 7 Number 4

Well, here it is, the month of May. When you read this, Gordon Bentzen will be in USA chatting with Colour Computer users over there, and getting as much information for us as he can. I am standing in for him this month. We are looking forward to hearing about new material, and new alliances with othe OS9 users around the world. Who knows what will eventuate? I think it may well be the start of a whole new era for the Australian OS9 users.

I continue to be surprised by calls from Colour Computers who want to become members of this Usergroup. Here I keep thinking that all the old users have got out of the CoCo scene, but no.... more pop up all the time. Great!

I know we have talked about this before, and I have seen the same comments on the OS9 echo on FIDO-Net, but here goes anyway. I would like to know what you people out there would like see in the way of software for OS9. Maybe it is already written, and we can tell you where to find it, either in the Public Domain, or as commercial software. If your wants are not too extravagant, maybe we will be able to write it ourselves. In fact I'm sure some of you will have done some of that already. Maybe you have written a script file that does something on your system? Please realise that other people can benefit from what you have done, even if it seems insignificant to you. Have a look at the article about script files in this issue, it gives some examples of how things can be done without writing special programmes to do them. I myself use script files for a lot of things. If I want to call up the Profile database programme, I use a script called 'PF' which changes directories for me, and starts up Profile. That way it is always at hand.

In this issue also, I have written a utility to rewrite a file starting from an offset within that file. Sometimes useful for cutting extraneous data from the front of a file. It is programmes like that which I love to write. Usually someone at a Usergroup meeting will say, how can I do this, or, why doesn't this programme allow this? This will usually get my ideas flowing hot again, and out pops a little utility programme. It doesn't matter that the utility has been done before, because usually I have learnt something in it's writing. Please ask me to solve your problems, it will help me to continue my learning of the C programming language. As always, if you haven't got the OS9 C compiler, the programme will become available in our PD library.

The last of the series 'A C Tutorial' is featured in this issue. It has been a long stretch, but I think it has been worth printing. While some of the comments in the series have been specifically MS-DOS oriented, most of the text could be applied to the OS9 C compiler. The complete archive, which includes the source code of the programmes mentioned, is available from our PD library. It is rather large, 198,400 bytes, so users with only single-sided disk drives would need to have it split up. I'm sure Jean-Pierre would do that if you ask hime nicely.

I would like to hear from you what you would like to feature in the Newsletter as a replacement. We have also received a PASCAL tutorial, but I have not really given much thought to using that for two reasons. 1. The tutorial is aimed at the MS-DOS Turbo-Pascal, and 2. OS9 Pascal is not very widely used, and is very out-of-date. Let me know what you think.

Regards,
Bob Devries

oooooooooooOOOOOOOOOOooooooooooo

Script2 - Some More Shell+ v2.0 Scripts
by Steve Clark

This file contains some more shell+ scripts for the new version of Shell+ for OS-9 Level II on the Color Computer. With the numerous capabilities provided by Ron Lammardo, Kevin Darling and Kent Myers, I offer these as suggestions only, you will need to modify them to your liking. In most cases, I am sure there are alternate ways to do what these scripts do. Some of these use other programs available on-line on various information services. I will try to mention these when they are encountered.

Batchfmt - Batch Formatting Floppies

If you need to format a series of floppies, say to use for backing up your hard disk, this shell script automates some of the process. You supply an initial name, and the script formats a floppy with "name00001" then asks if you want to do it again. By replying yes, the script prompts you to insert a new disk, etc. Drive /d0 is hard wired into the script, but you can change it (or add an option to ask which drive). It uses the GOTO and INC options from the new shell+.

```
*batchfmt
display c
load format
prompt Disk Name:  var.0
*loop
inc.1
display c
echo Place disk in drive /d0
prompt Press ENTER when ready to format %0%1:
var.2
format /d0 r "%0%1"
display 7 a
prompt Another (y/n):
if [ -y ]
  clrif
  goto loop
fi
unload format
echo Batchfmt Done.
```

Deskmate - Run Deskmate 3 Applications

This shell script runs deskmate 3 applications by presenting a menu and starting either the deskmate interface, or one of several deskmate applications. As with other scripts, you will have to use your own subdirectory names (I use /dd/usr/data/ss). It will set up a type 1 window and run the specified deskmate program in that window. See the script for further information.

```
*deskmate - deskmate execute
onerr goto +trap
display c
chd /dd/USR/DATA/SS
chx /dd/USR/DATA/SS/CMDS
path=/dd/CMDS
echo DeskMate
display a
echo 1 DeskMate Desktop
echo 2 Spreadsheet
echo 3 Word Processing
echo 4 Communications
display a
prompt Select:  var.0
goto +label%0
*label1
xmode /w6 type=1;display c>/w6 (desk<>>>/w6;xmode
/w6 type=80)&
goto +finis
*label2
xmode /w7 type=1;display c>/w7 (desk
dmledger<>>>/w7;xmode /w7 type=80)&
goto +finis
*label3
xmode /w8 type=1;display c>/w8 (desk dmtext<>>>/w8;
xmode /w8 type=80)&
goto +finis
*label4
xmode /w9 type=1;display c>/w9 (desk dmterm
termstat<>>>/w9;xmode /w9 type=80)&
*finis
display a
echo Task started in another window.
echo Use CLEAR key to change windows.
*trap
```

Playit - Play Sounds from a Menu

I found Kevin Darling's play program fascinating to use, and have collected several digitized sound files. The one thing I never can remember is what parameters to pass to each file. One way to handle this (or to have the computer remember for you) is to maintain a script file such as "playit" and put the parameters in. Again, specify your own directory structure.

```
* playit - play sounds
onerr goto +lab
load play
chd /dd/usr/sound
*repeat
echo 1 HAL from 2001        2 Late Night Breaking
Glass
```

```
echo 3 I'll be back        4 Captain Kirk
echo 5 Disruptor           6 Scotty
echo 7 Clint Eastwood      8 Laugh
echo 9 Monty Python
echo
prompt Which Sound (ENTER to Quit):  var.0
goto +lab%0
*lab1
  echo HAL - Can't Do
  play 18 cantdo.snd</1
  goto repeat
*lab2
  play 11 david1.pla</1
  goto repeat
*lab3
  play 28 back.mac</1
  goto repeat
*lab4
  play -29 kirk.pla</1
  goto repeat
*lab5
  play -28 dsrpt.pla</1
  goto repeat
*lab6
  play -28 scotty.pla</1
  goto repeat
*lab7
  play -11 clint.pla</1
  goto repeat
*lab8
  play 40 laugh.pla</1
  goto repeat
*lab9
  play -10 bing.pla</1
  goto repeat
*lab
unload play
```

Address - Build an Address File

This won't replace a database by any means, but it is a quick and dirty way to create a program to obtain prompted input from a user. It adds to an address file called "address.dat". You can use the same idea to create any type of ascii data file. It uses the IF, append, prompt, and GOTO features of the new shell+.

```
*address
var.0="address.dat"
*repeat
prompt Last Name:  var.1
prompt First Name:  var.2
prompt Address:  var.3
prompt City, State:  var.4
prompt Zip:  var.5
prompt Phone (999-999-9999):  var.6
prompt ------ Add to file (y/n):  var.9
if %9=y
  echo %1, %2>+%0
  echo %3>+%0
  echo %4 %5>+%0
  echo %6>+%0
  echo ->+%0
else
  echo Record not added.
fi
prompt Add Another (y/n):  var.9
if %9=y
  clrif
  goto repeat
else
  echo Done. %0 Data input complete.
fi
```

Man - Online Manuals

One of the nice features of having a hard disk is the ability to keep some things on-line which you would otherwise have to store on floppy or in the case of documentation, keep printed copies. I created this shell script to simulate the MAN command (somewhat) by providing access to documentation files. I have a subdirectory called /dd/usr/man which stores the text files I want to access, and use this script to choose the one I want to view. Pete Lyall's MORE command allows you to page back and forth within a file, and is perfect for implementing this application. When you execute man, something like the following appears:

```
     Directory of /dd/USR/MAN  00:11:48
     CLib              Datamod           MacPaint
Mail
     MaxiPic             Mkdir               NBS
Pilot
     Shell               TelStar             Tiny
UltiMuse
     Xcom9          Xlisp

    Which manual entry:
```

You then type the name at the prompt (or press ENTER to ignore) and the documentation is available. Note that the exact names will depend on what you have in the /dd/usr/man directory. This example shows what I have in mine at the present time. The man script looks like the following:

```
*man - manual processor
onerr goto repeat
cd /dd/usr/man
echo
dir /dd/USR/MAN
```

```
echo
*repeat
prompt Which manual entry:  var.0
if -r %0
  echo
  echo  Manual for %0:  Use SPACE/B to page through,
Q to quit
  /dd/cmds/more %0
else
  if %0 >= a
    clrif
    echo — man: No manual entry for %0
    goto repeat
  fi
fi
```

The "man" included in this archive is a datamod version, and will look slightly odd if you try to list it. If you want the pure ASCII, use Ron Lammardo's datamod or take it from this text file. It helps to keep the directory sorted (or you may want to use one of the "LS" type commands which print ordered directories).

Tyme - Digital Clock

We all seem fascinated with turning our expensive computers into cheap clocks, myself included. I have used the new shell, along with DATE, DISPLAY, and SLEEP to create an on-screen digital clock. It doesn't do anything but sit there and display the date and time in the middle of an 80 column window. Use it when you are going to leave your computer idle for a few minutes. To stop it, use control-E.

```
*tyme
onerr goto +trap
tmode -pause
display c 05 20
*repeat
display 2 3a 2b
date t
sleep 30
goto repeat
*trap
display c 05 21
date t
echo Tyme end.
```

Use these as supplied, or use them to generate ideas for your own shell+ scripts.

Steve Clark - Compuserve 73135,1204 <EOF>

oooooooooooOOOOOOOOOOooooooooo

Chapter 14 - Example Programs
WHY THIS CHAPTER?

Although every program in this tutorial has been a complete program, each one has also been a very small program intended to teach you some principle of programming in C. It would do you a disservice to leave you at that point without introducing you to a few larger programs to illustrate how to put together the constructs you have learned to create a major program. This chapter contains four programs of increasing complexity, each designed to take you into a higher plateau of programming, and each designed to be useful to you in some way.

DOSEX will illustrate how to make DOS system calls and will teach you, through self-study, how the system responds to the keyboard. WHATNEXT reads commands input on the command line and will aid you in setting up a variable batch file, one that requests an operator input and responds to the input by branching to a different part of the batch file.

LIST is the source code for the program you used to print out the C source files when you began studying C with the aid of this tutorial. Finally we come to VC, the Visual Calculator, which you should find to be a useful program even if you don't study its source code. VC uses most of the programming techniques we have studied in this course and a few that we never even mentioned such as separately compiled subroutines.

We will take a look at the example programs one at a time but without a complete explanation of any of them because you have been studying C for some time now and should be able to read and understand most of these programs on your own. One other thing must be mentioned, these programs use lots of nonstandard constructs and you will probably need to modify some of them to get them to compile with your particular compiler. That will be left as an exercise for you.

DOSEX.C - The DOS Example Program

The copy of DOS that you received with your IBM-PC or compatible has about 50 internal DOS calls that you can use as a programmer to control your peripheral devices and read information or status from them. Some of the earlier IBM DOS manuals, DOS 2.0 and earlier, have these calls listed in the back of the manual along with how to use them. Most of the manuals supplied with compatible computers make

no mention of these calls even though they are extremely useful. These calls can be accessed from nearly any programming language but they do require some initial study to learn how to use them. This program is intended to aid you in this study.

Display the program on your monitor or print it out for reference. It is merely a loop watching for a keyboard input or a change in the time. If either happens, it reacts accordingly. In line 23, the function "kbhit()" returns a value of 1 if a key has been hit but not yet read from the input buffer by the program. This is a nonstandard function and may require a name change for your particular compiler. There will probably be several similar calls that will need changed for your compiler in order to compile and run the programs in chapter 14.

Look at the function named "get_time" for an example of a DOS call. An interrupt 21(hex) is called after setting the AH register to 2C(hex) = 44(decimal). The time is returned in the CH, CL, and DH registers. Refer to the DOS call definitions in your copy of DOS. If the definitions are not included there, Peter Nortons book, "Programmers Guide to the IBM PC" is recommended as a good reference manual for these calls and many other programming techniques.

Another useful function is the "pos_cursor()" function that positions the cursor anywhere on the monitor that you desire by using a DOS interrupt. In this case, the interrupt used is 10(hex) which is the general monitor interrupt. This particular service is number 2 of about 10 different monitor services available. This particular function may not be needed by your compiler because some compilers have a cursor positioning function predefined for your use. This function is included here as another example to you.

The next function, service number 6 of interrupt 10(hex) is the window scroll service. It should be self explanatory. In this program, the cursor is positioned and some data is output to the monitor, then the cursor is "hidden" by moving it to line 26 which is not displayed. After you compile and run the program, you will notice that the cursor is not visible on the monitor. This is possible in any program, but be sure to put the cursor in view before returning to DOS because DOS does not like to have a "hidden" cursor and may do some strange things.

Some time spent studying this program will be valuable to you as it will reveal how the keyboard data is input to the computer. Especially of importance is how the special keys such as function keys, arrows, etc. are handled.

WHATNEXT.C - The Batch File Interrogator

This is an example of how to read the data on the command line following the function call. Notice that there are two variables listed within the parentheses following the main() call. The first variable is a count of words in the entire command line including the command itself and the second variable is a pointer to an array of pointers defining the actual words on the command line.

First the question on the command line, made up of some number of words, is displayed on the monitor and the program waits for the operator to hit a key. If the key hit is one of those in the last "word" of the group of words on the command line, the number of the character within the group is returned to the program where it can be tested with the "errorlevel" command in the batch file. You could use this technique to create a variable AUTOEXEC.BAT file or any other batch file can use this for a many way branch. Compile and run this file with TEST.BAT for an example of how it works in practice. You may find this technique useful in one of your batch files and you will almost certainly need to read in the command line parameters someday.

An interesting alternative would be for you to write a program named "WOULD.C" that would return a 1 if a "Y" or "y" were typed and a zero if any other key were hit. Then your batch file could have a line such as;

```
WOULD YOU LIKE TO USE THE ALTERNATIVE METHOD (Y/N)
```

Dos would use "WOULD" as the program name, ignore the rest of the statement except for displaying it on the screen. You would then respond to the question on the monitor with a single keyhit. Your batch file would then respond to the 1 or 0 returned and either run the alternative part of the batch file or the primary part whatever each part was.

```
WOULD YOU LIKE PRIMARY (Y/N)
IF ERRORLEVEL 1 GOTO PRIMARY
(secondary commands)
GOTO DONE
:PRIMARY
(primary commands)
:DONE
```

LIST.C - The Program Lister

This program is actually composed of two files, LIST.C and LISTF.C that must be separately compiled and linked together with your linker. There is nothing new here and you should have no trouble compiling and linking this program by reading the documentation supplied with your compiler.

The only thing that is new in this program is the inclusion of three "extern" variables in the LISTF.C listing. The only purpose for this is to tie these global variables to the main program and

tell the compiler that these are not new variables. The compiler will therefore not generate any new storage space for them but simply use their names during the compile process. At link time, the linker will get their actual storage locations from the LIST.OBJ file and use those locations for the variables in the LISTF part of the memory map also. The variables of those names in both files are therefore the same identical variables and can be used just as any other global variables could be used if both parts of the program were in one file.

There is no reason why the variables couldn't have been defined in the LISTF.C part of the program and declared as "extern" in the LIST.C part. Some of the variables could have been defined in one and some in the other. It is merely a matter of personal taste. Carried to an extreme, all of the variables could have been defined in a third file and named "extern" in both of these files. The third file would then be compiled and included in the linking process.

It would be to your advantage to compile, link, and run this program to prepare you for the next program which is composed of 5 separate files which must all work together. VC.C - The Visual Calculator This program finally ties nearly everything together because it uses nearly every concept covered in the entire tutorial. It is so big that I will not even try to cover the finer points of its operation. Only a few of the more important points will be discussed.

The first thing you should do is go through the tutorial for VC included in the file VC.DOC. There are several dozen steps for you to execute, with each step illustrating some aspect of the Visual Calculator. You will get a good feel for what it is capable of doing and make your study of the source code very profitable. In addition, you will probably find many ways to use the Visual Calculator to solve problems involving calculations where the simplicity of the problem at hand does not warrant writing a program.

Notice that the structure definitions, used in all of the separate parts of the program, are defined in the file STRUCT.DEF. During program development, when it became necessary to change one of the structures slightly, it was not necessary to change it in all of the files, only one file required modification which was then "included" in the source files. Notice that the transcript data is stored in a doubly linked list with the data itself being stored in a separate dynamically allocated char string. This line is pointed to by the pointer "lineloc".

For ease of development, the similar functions were grouped together and compiled separately. Thus, all of the functions involving the monitor were included in the file named VIDEO.C, and all of the functions involving the data storage were grouped into the FILE.C collection. Dividing your program in a way similar to this should simplify debugging and future modifications.

Of special interest is the "monitor()" function. This function examines the video mode through use of a DOS command and if it is a 7, it assumes it is a monochrome monitor, otherwise it assumes a color monitor. The colors of the various fields are established at this time and used throughout the program. Most of the data is written directly to the video memory, but some is written through the standard BIOS routines.

The file DEFIN.C is simply a catalogue of the functions to aid in finding the functions. This file was generated as one of the first files and was maintained and updated for use during the entire design and coding lifetime.

Feel free, after understanding this code, to modify it in any way you desire for your own use.

oooooooooOOOOOOOOOOooooooooo

OZ - OS9 BBS

by Rod Holden - Sysop

Hi, this is your Sysop once again letting you know how the BBS is coming along and what sort of info is available.

The King James bible is now unpacked. I had to be careful when unpacking it because for instance when you unpack John and then unpack John_1, John_1 will write over the top of John, because the filenames had spaces in them. To get around this problem you must first unpack John_1 which will show as John then rename that file to John_1 or John.1 or how ever you like to rename that file. Sorry if anyone was misled in the newsletter which mentioned about the bible scanner. It is currently being written so stay tuned to this station.

In the OS9_UTI (OS9 Utilities) directory there is a file called FIXTXT.AR which has two programmes called Fixtxt and Lf2Cr. Here is the doc file about it:

Fixtxt

Use: Fixtxt path
 or: Fixtxt <path1 >path2

Strips linefeeds and other control codes, fixes backspaces, tabs, and removes trailing spaces. A large buffer is used for significant time savings over similar programs.

Lf2Cr

Use: lf2cr path
 or: lf2cr <path1 >path2

Similar to fixtxt, except it replaces linefeed with carriage returns.

Warning: The non-filter mode of these programs will modify the file. Use ONLY on text files, and use a backup if the text is important!

If anyone in the user group has software that is PD (that is not in the PD library and is not commercial software) feel free to upload it to the BBS or send a copy to Bob Devries or Jean-Pierre Jacquet (your disk will be returned) so other users can obtain a copy.
I said in the newsletter that I was adding a second HD to my system as I was running out of space for the BBS, I now have that second HD which gives a total space of 90MB.

Logging On.
Please contact me before logging on so I can enter your name in the userlog. Then, when you 'log on' to the BBS type your name (eg: Rod Holden), press enter, then press enter at the password prompt. If 'setup' is available to you, go into that area and set the option for number of lines per page (24 is what most users set) and also the option for change of password and enter your password.

Well that's it from me till next time see you in the bit stream.

Your Sysop
Rod Holden

PS. New times for OZ - OS9 BBS are 2000 - 2230hrs (AEST)

oooooooooooOOOOOOOOOOooooooooooo

Rewrite, A utility to strip the front from a file
by Bob Devries

I recently had the need to strip some garbage characters from the front of an otherwise perfectly good file. As usual, I had to write it myself. Here is the source in C. I first wrote it for use on my Amiga 2000, using SAS/C, which is ANSI compatible, and then copied it to the Colour Computer under OS9, and, yep, you guessed it, it didn't work, even when I had converted what I thought was all the necessary parts.
It seems that the OS9 C compiler is quite different in quite a few areas. Firstly, I used the function 'sscanf' to convert the ASCII command line into a long variable. I came unstuck there. On the Amiga, I had written 'sscanf(argptr,"%lx",&offset)', and left the '0x' on the front of the command line variable. This does not work for CoCo OS9 C. As you can see, I had to remove the '0x', and change the function call to 'sscanf(argptr,"%X",&offset)'. Notice the capitalisation of the '%X'. Refer to the C manual on page 4-26 (as I should have done - it doesn't pay to be too sure of yourself). In the Amiga version, I also added the '0x' instead of the user supplied '$'. This was also unnecessary. Ok, so after a bit of trial and error, here is the code.

Bob Devries

```
/* rewrite - (c) 1993 Bob Devries   */
/* rewrites the specified file from */
/* the specified offset. Offset may */
/* be either decimal or hexadecimal */
/* e.g. 0x498 == $498 == 1176       */

#include <stdio.h>
#include <string.h>

#define TRUE 1

main(argc, argv)
int argc;
char *argv[];
```

```
{
    FILE *in, *out, *fopen();
    int ch;
    char temp[32];
    char *argptr;
    long offset = 0L;
    long atol();

    argptr = argv[3];               /* point to the offset        */

    if (argc != 4) {                /* oops, user error           */
         fprintf(stderr,"Usage: %s <infile> <outfile> <offset>\n",argv[0]);
         fprintf(stderr,"       <offset> may be either decimal or\n");
         fprintf(stderr,"       hexadecimal. (e.g. 0x498 or $498)\n");
         exit(0);
    }

    if ((in = fopen(argv[1],"r")) == NULL) {
         fprintf(stderr,"Can't open %s for input.\n",argv[1]);
         exit(errno);
    }

    if ((out = fopen(argv[2],"w")) == NULL) {
         fprintf(stderr,"Can't open %s for output.\n",argv[2]);
         exit(errno);
    }

    if ((argv[3][0] == '0') && (toupper(argv[3][1]) == 'X')) {
         argptr += 2;               /* increment past the 0x      */
         sscanf(argptr,"%X",&offset);
    } else if (argv[3][0] == '$') {
         strcpy(temp,++argptr);     /* increment past the $ sign  */
         sscanf(temp,"%X",&offset); /* the %X must be capitalised */
    } else {
         offset = atol(argv[3]);
    }

    fseek(in,offset,0);

    while (TRUE) {
         if ((ch = getc(in)) == EOF) break;
         putc(ch,out);
    }
    fclose(in);
    fclose(out);
}

/* EOF */
```

oooooooooOOOOOOOOOooooooooo

Conversation of Interest/Help
by Rod Holden

Here is a conversation that took place between myself and Bob Devries.
We were talking about how long OS9 has been available and running. And I was amazed (after inheriting a collection of 'The Rainbow' dating back to 1983) at reading some of the articles. I said to

Bob that I feel rather cheated because I wasn't aware that OS9 had been going for so long. I bought my CoCo3 back in 1988 and I learned about OS9 was in 1989. I said to Bob that I will never be able to make up for lost time, not to worry we all feel the same way at one stage or another. I said my expertise on OS9 would fit on the head of a pin, so to all you users out there, here I am running the National OS9 User Group BBS with limited knowledge of OS9. So if you feel like asking questions no matter how big or small or silly please contact us in Brisbane by letter or by logging on to the BBS, or by phone and we will try very hard to answer your questions.

OS9 Level II is a system that shocks those users who have been using it for a long, and I can still hear them say "why did it do that" or "the instructions said it is supposed to do this" but it didn't why not?? Do you sit there for hours scratching your head reading the instructions over and over again till you either give up in disgust or throw the instructions away and you try something different and find it works, or do you ring someone for help, or forget all about it because asking makes you feel like an idiot. Does this sound like you, if so don't dispair, as I said earlier help is around the corner. We await your questions.

bye
Rod Holden

oooooooooooOOOOOOOOOOooooooooooo

Members' Questions and Answers

Brian Palmer, of Balgownie, NSW asks:

I would like to know if there is any way to use the ORCHESTRA 90CC cartridge under OS9, and also if there is a way to piggy-back the disused 128K RAM chips, on top of the already installed 512K RAM board.

ANSWER 1. I see no reason why the ORCHESTRA 90CC cartridge could not be used under OS9, however, as far as I know, no-one has as yet done so. If someone wrote a programme to use it, it would have to be written like Kevin Darling's 'PLAY' command, which plays music through the monitor speaker. The main problem with this programme, is that it stops the interrupts, thus not allowing multi-tasking, which is what OS9 is all about.

ANSWER 2. No, the chips removed when you upgrade to 512K are of a different type than those used in the 512K board, and must not be piggy-backed on them. Perhaps if someone who is clever in electronic design could have a look at the circuit diagram for the RAM board in 'The Rainbow', by Tony Distafano some time back, they could perhaps be used there. How about it someone? I know there are some electronics experts down Melbourne way!

Hope this helps,

Bob Devries.

oooooooooooOOOOOOOOOOooooooooooo