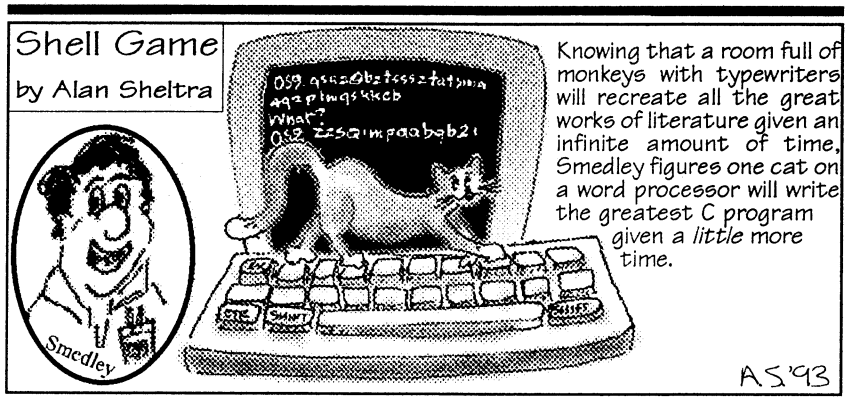
Volume One, Issue Eleven

$3.00

# The "International" OS9 Underground®

A Fat Cat® Publication

Magazine Dedicated to OS-9/OSK Users Everywhere!




***Coming Soon...* "Shell Game" OS9 Underground 1994 Calendar**

## Advertisers Index



**Let these fine vendors know you saw it in The Underground!**



# The "International" OS9 Underground® Magazine

Dedicated to OS-9/OSK Users Everywhere

Volume 1, Issue 11

## CONTENTS





### The Underground Staff

**Editor/Publisher:**
Alan Sheltra

**Assistant Editors:**
Jim Vestal
Steve Secord

**Technical Editor:**
Leonard Cassady

**Contributing Editors:**
Wayne Campbell
Mike Guzzi
Allen Huffman
Scott McGee
Eric Levinson
Boisy Pitre
Bob van der Poel

**Typesetting/Layout/Artwork:**
AniMajik Productions

**Proofing:** Leonard Cassady

# Mail Room

## Any "DiskMaster's" out there?

*Enjoyed "OS9 Underground" mensely; keep up the good work.*

*Does anyone out there have one of the D.P. Johnson's DiskMasters? God I hope I am not the only one who bought the thing. I would like to contact [other] users. Also, do you have any info on the Level 2 versions. I have V2.01. I see reference to V2.4. Am I missing something?*

*-Leonard H. Reed, Sr., Gold Hill, OR.*

I know of 2 DiskMaster user's, one in Canada, (Dieter Rossmann), and one in your own home state, (Paul Pollock). I'm sure there are others out there as well.

The version V2.4 you are referring to is for OS-9/68K, (OSK). The last, (released), version for the CoCo is the one you have.

-Editor

## Feedback on Proposed Magazine Changes...

[In response to the reader survey card]

*That special binding you're considering will likely cause the envelope to tear. Why not consider 2 color printing and "real" typesetting and offset printing?*

*-Bruce Moore, Chantilly, VA*

Bruce, since you are in the printing business you might be familiar with "comb-binding". This binding process is very lightweight and will not tear the envelope no matter how rough the Post Office handles it. I have used this method before and never had a document, bound in this fashion, tear an envelope.

While I would love to send my output to a linotronic typesetter (1200 to 2400 DPI), and add two color printing, the cost would be extremely prohibitive. Unfortunately, the subscriber base does not yet make those "niceties" cost effective as yet. Someday though!

-Editor

## Ooops...

*From: PHXKEN@delphi.com*
*Subj: autofmt_OS9Underground*
*To: ZOGster (On Delphi)*

*Alan, Paul Fitch has helped me on FIDO to make the basic09 program on page 16 work correctly. Several spaces were not shown on the SHELL call line of autofmt. There should spaces before and after the "r" and there seems to be a space missing after the word Format. The line should read:*

```
SHELL "Format "+drivename+" r "+
      CHR$(34)+diskname+CHR$(34)
```

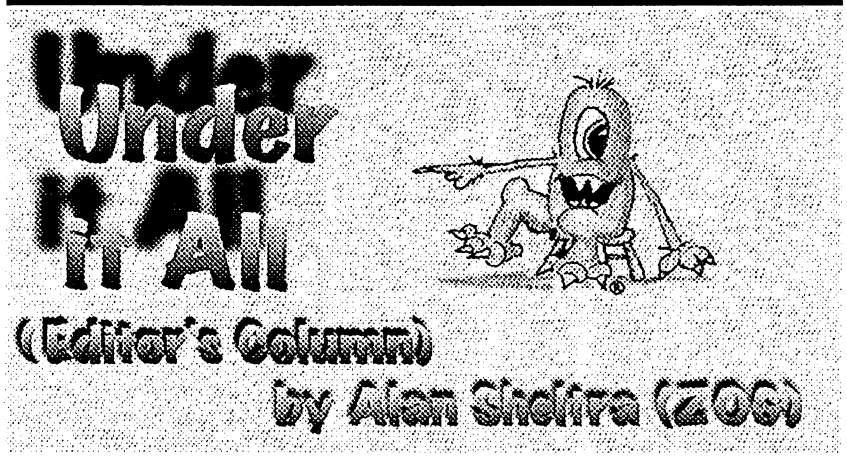
*-Farrell Kenimer alias Ken/Az*

Thank you for pointing out that error, Ken. This was an error, most likely, created by *me* rather than the author though.

-Editor

Letters to the Editor may be sent to:
4650 Cahuenga Blvd., Ste #7
Toluca Lake, CA 91602
or as email to the following addresses:
EditorOS9U@AOL.com
-or-
OS9Under@AOL.com or
to "TheFatCat" Mail List
"TheFatCat@AOL.com" (See Page 9)
"ZOGster" (on Delphi)

```
LINEFEED STX POINTER
 LDA #StdOut
 LEAX LF,PCR
 LDY #2
 OS9 I$Write      output carraige return/line feed
 LDX POINTER

 LBRA EXEC        go back to start...

* Display help message.

HELP LDA #StdOut   1=std out
 LEAX HELPMSG,PCR  point X to help message
 LDY #helpsize     load Y with length of help message
 OS9 I$Write       write help message to screen
 BRA EXIT          goto EXIT

* Get prompt text.

GETPROMPT
 CLR COUNT,U       clear COUNTer
 STX PMTPTR        store location of prompt text in
PMTPTR
GLOOP LDA ,X+      X points to command line, load
character in A
 CMPA #'"          is it a quote?
 BEQ GEND          if yes, goto GEND
 CMPA #$D          is it enter?
 BEQ GEND          if yes, goto GEND
 INC COUNT,U       increment COUNTer
 BRA GLOOP         goto GLOOP
GEND LDA COUNT     load COUNT into A
 INC COUNT,U       increment it (0-9 becomes 1-10,
basically...)
 STA PMTLEN        store length of prompt in PMTLEN
 LBRA EXEC         goto EXEC

* Return to OS9.

EXIT CLRB
ERROR STB COUNT    store error code in temporary
COUNTer

 LEAX BUFFER,U     point X to tmode buffer
 LDA ECHO          load A with original echo status
 STA +4,X          restore it
 LDA PAUSE         load A with original pause status
 STA +7,X          restore it
 LDA #StdIn        0=std in (current path)
 CLRB              0=ss.opt
 OS9 I$SetStt      restore tmode settings

 LDB COUNT         load B with saved error code
 OS9 F$Exit        then exit

PROMPT
 BSR PROMPTER      output prompt
 LDX POINTER       point X to command line position
 LDA -1,X          decrement (go back on character)
 CMPA #$D          is it ENTER?
 LBEQ EXIT         if so, done...
 LBRA EXEC         else go back to start

* Display prompt, wait for keypress, eradicate prompt,
format hard drive.

PROMPTER
 LDB PMTLEN        load B with how many characters
to output
 STB COUNT         store it in COUNTer (to be used in
a moment)
 CLRA              A=0 (building D register from A/B, D=rArB)
 TFR D,Y           transfer D to Y register
 LDA #StdOut       1=std out
 LDX PMTPTR        point X to prompt text
 OS9 I$Write       write prompt to screen
 BCS ERROR         if error, goto ERROR

* Read one character.

 CLRA              0=std in
 LEAX LINEBUF,U    point X to line buffer
 LDY #1            load Y with max. # of characters to read
 OS9 I$Read        read character
 BCS ERROR         if error, goto ERROR

* Backspace over whatever prompt was used.

 LDA #StdOut       std out
 LEAX BACKSPACE,PCR point X to backspace text
 LDY #3            load Y with max. # of characters to output
LOOP OS9 I$Write   write backspace text to screen
 BCS ERROR         if error, goto ERROR
 DEC COUNT         decrement COUNTer
 BNE LOOP          if COUNTer not 0, goto LOOP
 RTS

PROMPTLEN FCB 22
PROMPTMSG FCC "Press [Enter/Return) :....."
BACKSPACE FCB $08,$20,$08
 FILEMESS FCC "[Press [A)bort, [Q)uit, or [S)top to
Abort File]"
 FCB $D,$A
LF FCB $D,$A
filesize equ *-FILEMESS
HELPMSG FCB $D,$A
 FCC "SEListor 1.05 - By Allen Huffman of Sub-Etha Software."
 FCB $D,$A
 FCC "Syntax: SEListor "Prompt" filename1 {"prompt2"} {filename2}
..."
 FCB $D,$A
 FCC "Usage : Displays text file(s) with page pausing
prompt, which is"
 FCB $D,$A
 FCC "    changeable. Also functions as a stand-alone
'pause' command."
 FCB $D,$A
 FCC "    Multiple files and/or prompts may be used."
 FCB $D,$A
helpsize equ *-HELPMSG

 ENDSECT
```




## Rocket Crash

Our lead story, in issue 8, as some of you may recall, was "The Rocket, but will it Fly?". Well, it seems it's destined not to. Here's an excerpt of a message reply from Chris Burke on Internet:

[Zack Sessions notes:]
>As I understand it, one of the main reasons Chris
>decided to not produce the Rocket was that
>Microware screwed him on the OSK license quote.

Ouch! As noted in a previous posting, there are no villians here. Microware changed their product line, and with it their pricing. The new pricing made it significantly less attractive for Burke & Burke to enter the OSK hardware market with a product which would ship only 100 units. There are some real advantages to the new pricing, but in our case it essentially doubled our production costs. Depending on our ship date, we might have been able to get the older Microware pricing and product. Rocket orders never reached the 100 unitmark, so Burke & Burke couldn't commit to an order. Microware was both flexible and generous, but the deal just didn't work out.

Chris made an honest effort with the "Rocket" board for the CoCo and it's really a shame things did not work out as planned. It would have given the CoCo a much needed kick in the pants.

## DELMAR Announces Port of GWindows to the MM/1

Ed Gresick, of DELMAR CO., asks:
*"Are you interested in G-WINDOWS for the MM/1? If so, I am willing to do a port of G-WINDOWS to the MM/1 Computer. Price will be $200.00 per copy."*
*"This will include the software, all the G-WINDOWS utilities, demo programs, images, and sample source code. Documentation includes the G-WINDOWS manual and a tutorial prepared by DELMAR CO."*
*"Before starting the port, I will require twenty (20) confirmed orders. At this point, I have received 8 confirmed orders. After I receive the necessary commitments, 2 to 3 months will be required to do the port. Orders may be made by mail or e-mail accompanied by check or credit card information. Checks and credit card payments will be held until 20 orders are received or December 31, 1993. If the necessary orders are not received by December 31, 1993, checks will be returned and credit card payments not processed.*

Call Ed at (302) 378-2555 for more info. If you have an MM/1.. go for it! I have been using GWindows on my TC70 for over a year now and really enjoy it.

## Outta Space... Again!

Returning next month is Eric Levinson's column as well as our BBS Listing we had hoped to fit in this issue. Also, Allen Huffman's COMPLETE Atlanta Fest Report! C'ya next month! -Alan Sheltra (ZOG)





```
STA PMTLEN,U      store it in PMTLEN
LEAX PROMPTMSG,PCR point X to default prompt
STX PMTPTR,U      store it in PMTPTR

LDA #StdIn        0=std in
CLRB              0=ss.opt function
LEAX BUFFER,U     point X to buffer
OS9 I$GetStt      read tmode settings into BUFFER
LBCS ERROR        if error, goto ERROR
LDA +4,X          load A with current echo status
STA ECHO          save it for later in ECHO
LDA +7,X          load A with current pause status
STA PAUSE         save it for later, also, in PAUSE
CLRA              set A to 0
STA +4,X          turn off echo
STA +7,X          ...and pause
LDA +8,X          load A with lines per page
DECA              decrement A (make room for PAUSE msg on screen)
STA PAGELEN       store it in PAGELEN
LDA #StdIn        0=std in (current path) (again)
CLRB              0=ss.opt function (also again)
OS9 I$SetStt      update status

LDX POINTER       restore X from POINTER

* Scan command line for options/parameters.

EXEC LDA ,X+      X points to command line, load char in A
CMPA #32          is it a space?
BEQ EXEC          if yes, goto EXEC and keep scanning
STX POINTER       save X in POINTER
CMPA #$D          is it enter?
LBEQ PROMPT       if yes, goto PROMPT
CMPA #'-          is it a dash?
LBEQ HELP         if yes, goto HELP
CMPA #'"          is it a quote?
LBEQ GETPROMPT    if yes, goto GETPROMPT

* If here, assume it must be a filename.

LEAX -1,X         decrement X {BRUTE FORCE!!!}
LDA #1            1=read acces
OS9 I$Open        open filename pointed to by X
LBCS ERROR        if error, goto ERROR
STA PATH          store path in PATH
STX POINTER       store new command line ptr
*                 position in POINTER
LDA PAGELEN       load A with page length
STA COUNT         store it in COUNTer

LDA #StdOut       Display abort message...
LEAX FILEMESS,PCR point to file message
LDY #filesize     length of message
OS9 I$Write       output it
DEC COUNT,U       decrement count twice to
compensate for
DEC COUNT,U       two lines of text already printed

LIST LDA PATH     load path into A
LEAX LINEBUF,U    point X to line buffer
LDY #80           load Y with max. # of chars to read
OS9 I$ReadLn      read line
BCS LERROR        if error, goto LERROR
LDA #StdOut       1=std out
OS9 I$WritLn      write line to std out
LBCS ERROR        if error, goto ERROR

LDA #StdIn        standard input
LDB #1            _gs_rdy (character ready syscall)
OS9 I$GetStt      check to see if character ready...
BCS LCONT         if nothing there, continue
LEAX LINEBUF,U    point X to line buffer
LDY #1            1 character to read
OS9 I$Read        read it
LDA LINEBUF       load A with character gotten at prompt
ANDA #$DF         make upper case
CMPA #'A          is it Abort?
BEQ LABORT
CMPA #'Q          is it Quit?
BEQ LABORT
CMPA #'S          is it Stop?
BEQ LABORT

LCONT
DEC COUNT         decrement COUNTer
BNE LIST          if not 0, continue at LIST

LBSR PROMPTER     display prompt
LDA LINEBUF       load A with character gotten at
prompt
ANDA #$DF         make upper case
CMPA #'A          is it Abort?
BEQ LABORT
CMPA #'Q          is it Quit?
BEQ LABORT
CMPA #'S          is it Stop?
BEQ LABORT
LDA PAGELEN       load A with page length
STA COUNT         store it in COUNTer
BRA LIST          goto LIST

LABORT LDA PATH   get filepath
OS9 I$Close       close it
BCS ERROR
LDX POINTER       restore command line position
LDA ,X            get character
CMPA #$D          are we at end of line?
LBNE EXEC         no, so continue scanning line
BRA EXIT          otherwise done.

LERROR CMPB #E$EOF is error code "end of file"?
BNE EXIT          if no, goto EXIT
LDA PATH          load path into A
OS9 I$Close       close the file
BCS ERROR         if error, goto ERROR

LDX POINTER       this is some redundant code, but
I think I must
LDA ,X            do it this way. IE, {BRUTE FORCE!!!}
CMPA #$D
BNE PROMPT
```

# LISTing in ASM

**by Allen C. Huffman**

When I was running an OS-9 BBS system, I decided I needed a more powerful way to display text files to my users than the normal "LIST" command. SEListers is the result. SEListers is a replacement for the normal "LIST" command but with some interesting modifications: It pauses the listing file at the end of each screen, it allows the user to abort the file while it is listing, and it gives a settable prompt to use either by itself or as the "end of screen" prompt when listing files. It will also handle multiple files in the sequence that you enter them on the command line. Here are some examples:

```
selister filename.txt
```

The above lists the file, pausing at end of each screen with a default prompt.

```
selister "[PAUSED]" filename.txt
```

This lists the file, pausing at the end of each screen using message "[PAUSED]".

```
selister "<Hit a Key>" filename.txt "<Press ENTER>"
```

This lists the file using "<Hit a Key>" as the screen pause prompt, then prompts with "<Press ENTER>" after the file has been displayed.

```
selister
```

By itself it displays a default "Press [Enter/Return) :" message.

```
selister filename1.txt filename2.txt
  "<PAUSE>" filename3.txt
```

shows the first two files using the default prompt, then shows the last file using the "<PAUSE>" prompt.

Everything else should be fairly self-explanatory. **"selister -?"** will display a help message. RMA source code is provided, as well as a BASIC09 program that will create the executable file. Extra space was left after the default prompt text to allc patching a different default prompt (up to 32 characters). If this is done, the first byte before the prompt text must be changed to the number of characters in the prompt. Enjoy!

Note: This routine is Copyright (C) 1993 by Allen Huffman of Sub-Etha Software. Use it as you wish, but please do not distribute modified versions without the author's consent.

**-Allen C.Huffman**

```
* SEListers 1.05 by Allen C. Huffman
* Copyright (C) 1993 by Sub-Etha Software
*
* Single/multiple file lister/prompter utility thing.
*
* 0.00 08/26/92 - initial version
* 0.01 06/27/92
* 0.02 08/31/92
* 0.03 09/10/92
* 0.04 10/22/92
* 1.04 11/21/92 - scan for "abort" after each line
* 1.05 07/17/93 - minor code change, update usage message

StdIn equ 0
StdOut equ 1
StdErr equ 2

 PSECT SEListers,$11,$81,0,200,START

 VSECT
PATH RMB 1
COUNT RMB 1
PAGELEN RMB 1
ECHO RMB 1
PAUSE RMB 1
PMTLEN RMB 1
PMTPTR RMB 2
POINTER RMB 2
LINEBUF RMB 80
BUFFER RMB 32
 ENDSECT

* Setup. Point to default prompt. Turn off echo and
pause, etc...

START
 STX POINTER      save X in POINTER
LDA PROMPTLEN,PCR load A with length of prompt
```

# Dual'n RS232 Ports

## by Richard Albers

As faster modems have become more affordable, they have become more and more popular. Fast modems are replaced with even faster ones, and the prices of the "old, slow" ones can be expected to drop. "Old" and "slow" are, of course, relative terms. The first serial circuits I worked on were "new and fast" 60 baud! Yes, 60; up from 50. A very few years ago 9600 was state of the art. Soon mega-bauds will abound. CoCo may not go quite that fast, but it can do 9600 or more. If you want to upgrade a Tandy Direct Connect Modem Pak(DCM) to more than 300 baud, and maybe with two ports, this is for you. I will describe how to remove the parts that make up the DCM's 300 baud-only modem and substitute new parts to interface your CoCo to an external 300 to 19200 baud modem, and optionally, add a second serial port with the same capabilities. You can build one port now and add a second port later. If you can solder small parts, use common hand tools, and follow instructions, you can convert a DCM pak to work like the popular Deluxe RS232 pak. While this is not a job for a beginner hacker, it makes a good intermediate level project.

Read the instructions several times until you understand all the steps. If you feel this project is too advanced for you, perhaps you can find someone to do the parts you don't want to do, in exchange for doing parts of their conversion. This could even be a club construction project.

You will need a few basic tools: Screwdrivers - a #1 Phillips and a small fl blade, small needle-nose pliers Diagonal cutter (flush-cutting is nice), large n clipper (straight cutting edges are best), a sharp knife, such as a Stanley utility knife, a small file and/or sandpaper A wire wrapper/unwrapper/stripper, a 25 watt and a 40 watt or controlled temperature soldering iron, small diameter solder (about 1/32 inch diameter), vacuum device or braid for removing solder, a small drill with 5/32" and 1/16" or smaller bits, a DVM (best), VOM, or continuity tester. These are the minimum; no hacker ever has too many tools!

### Some new parts - for dual ports:

2 - AT type serial cables
(with DB9S and DB25P connectors)
2 - Right angle PCB mount DB9P connectors to fit cables
1 6551A ACIA and
2 - 1488 and 2 - 1489 ICs
1 - 1.8432 MHz crystal similar to X1 in the DCM.

### For one port:

1 - AT type serial cable (with DB9S and DB25P connectors)
1 - Right angle PCB mount DB9P connector to fit cable
1 - 1488 IC
1 - 1489 IC

*Note:* The PCB mount DB9P connectors must have exposed conductors between the connector end and the PCB end. That's where the wires will connect. Get cables with the narrowest ends you can find. They should be no wider than the connectors they will plug into on the 9 pin end. You can substitute RJ45 (8 pin telephone type) connectors for the DB types. Get a cable first, then wire the RJ45 to put the signals on the correct pins of the





```
*  move.b VOut1(a3),(a0)+ move byte into (a0) and increment
   move.w d2,d1  move computed driver delay into d1
RecDelay subi.w #1,d1   subtract 1 from delay value
   bne.s RecDelay
   subi.l #2,RSd2(a5)
   bhs.s RecLoop   subtract 1 from buffer size
   clr.w d1    return SUCCESS
   rts     return to caller

Unknown: move.w #ESUnkSvc,d1  Unknown service code
   ori.b #Carry,ccr   return Carry set
GetSta99: rts
```

<End of Figure 1>

Figure 2

```
*******************
* SetStat:  set device status
*
* Passed:  (a1) = Path Descriptor
*          (a2) = Static Storage address
*          (a4) = process descriptor
*          (a5) = caller's register stack ptr
*          (a6) = system global data ptr
*          d0.w = status call function code
*
* Returns:  varies with function code
*
* Error Return:  (cc) = carry set
*                d1.w = error code
*
* SS_Play parameters:
*
*   a0.l = Pointer to buffer
* d0.w = Path number (unused)
* d1.w = Function code
* d2.l = Buffer size
* d3.w = Sample rate
*
SetStat:
   cmpi.w #SS_Play,d0   is it the play option?
   bne.s SetStat10   no, check next possibility
   move.l RSd3(a5),d2   move sample rate into d2.l
   divu.w #LoopConst,d2  divide by loop constant
   and.l #$FFFF,d2        throw away upper 16 bits
   movea RSa0(a5),a0   move buffer ptr into a0
   movea V_PORT(a2),a3  move base A/D address into a3
PlayLoop move.b  (a0)+,VOut0(a3)   move byte into
channel 0
   move.b (a0)+,VOut1(a3)  move byte into channel 1
   move.w d2,d1   move computed driver delay into d1
PlayDelay subi.w #1,d1    subract 1 from d1
   bne.s PlayDelay   loop if not zero
   subi.l #2,RSd2(a5)   subtract 2 from buffer size
   bpl.s PlayLoop   branch if buffer > -1
SetStatOK clr.w  d1     return SUCCESS
PutSta99: rts
```

```
* Dummy functions to appease the kernel
SetStat10 cmpi.w #SS_Open,d0   check for Open
   beq.s SetStatOK
   cmpi.w #SS_Relea,d0  check for Release
   beq.s SetStatOK
   cmpi.w #SS_Close,d0  check for Close
   beq.s SetStatOK
   bra.s Unknown
```

<End of Figure 2>

-Boisy G. Pitre

(LOF)

into play (such as caching) and even the type of processor you're running. For my 20MHz 68030, I've chosen a delay of 110.

## THE GETSTAT ROUTINE

Figure 1 shows the source code for the GetStat call. The SS_Record parameters attempt to follow the standard GetStat routine by providing for d0.w as the path number (although it isn't even referenced by the routine). a0.l holds the pointer to a pre-allocated buffer, d1.w has the function code SS_Record, d2.l holds the size of the buffer who's pointer was passed in a0, and d3.w holds the sampling delay in hertz (e.g 22KHz = 22000 in d3.w)

Upon entering the routine, we compare the SS.Record value with the value in register d0.w (which holds the status call function code). If another function code was passed, a branch to Unknown: is made, and the routine exits with an "Unknown Service Request" error code.

If the function code is SS_Record, we move the sample rate into d2. and divide that value by the loop constant. The resulting value is then ANDed with $FFFF to throw away the upper 16 bits of the long word. Now d2.l contains the loop delay which will be used by the driver when recording sound.

The caller's address buffer is then moved into a3 and bytes are subsequently moved from the A/D port into the buffer while a0 is incremented to accomodate the next byte. We then move the driver delay we computed earlier into d1, and subtract 1 from that value until we reach zero.

We then subtract 1 from the buffer size and loop back to reading from the A/D port until the size reaches zero (meaning we have reached the end of the buffer).

## THE SETSTAT ROUTINE

The SetStat routine is similar to the GetStat except that a pointer to valid data is handed to the driver routine and is passed to the A/D port.

Notice that this routine also contains "dummy functions" which are called at various intervals by the kernel. SS_Open, SS_Relea, and SS_Close are called by the kernel to open, release and close a device. This used by drivers to manipulate baud rate and parity parameters. Since our driver doesn't use them, we just clear d1 and return normally.

## IN CONCLUSION

Figure 3* is the driver in its entirety. While there are many issues that remain open, (such as interrupt service routines and minimal GetStat/SetStat calls), I hope you've gained some insight on the basics of OS-9 driver writing.

*Editor's Note: Due to space Figure 3 will appear in the Dec. Issue.

Figure 1.

```
*********************
* GetStat: get device status
*
* Passed: (a1) = Path Descriptor
*         (a2) = Static Storage address
*         (a4) = process descriptor
*         (a5) = caller's register stack ptr
*         (a6) = system global data ptr
*         d0.w = status call function code
*
* Returns: varies with function code
*
* Error Return: (cc) = carry set
*               d1.w = error code
*
* SS_Record parameters:
*
*   a0.l = Pointer to buffer
* d0.w = Path number (unused)
* d1.w = Function code
* d2.l = Buffer size
* d3.w = Sampling delay
*
GetStat:
  cmpi.w #SS_Record,d0  is it the record option?
  bne.s Unknown    no, exit w/ error
  move.l R$d3(a5),d2   move sample rate into d2.l
  divu.w #LoopConst,d2  divide by loop constant
  and.l #$FFFF,d2        throw away upper 16 bits
  movea R$a0(a5),a0   move caller's a0 (buff ptr) into a0
  movea V_PORT(a2),a3  move base A/D address into a0
RecLoop  move.b VOut0(a3),(a0)+  move byte into
*                                  (a0) and increment
```





DB25 connector to the modem. You can wire one end of the cable to a DB25S (cheaper) or use an RJ45 to DB25 adapter (easier). There are no standards for using RJ45 connectors here, you get to set your own. Just make both cables, (ports), the same. I used glue and cable ties to fasten the RJ45s to the PCB.

30 Gauge wire (several colors preferred) Hardware to mount connectors to PC board Epoxy or instant glue (crazy glue), and of course, a DCM pak to modify and a modem or two.

The parts for a dual port conversion almost total the minimum order of $30 from Jameco (1-800-831-4242). If you shop around, you may reduce the expense by half.

You should already have a CoCo, disk drive and Multi-Pak or equivalent with which to use your modified DCM pak.

Some warnings: No one guarantees your success. The author and staff of this magazine have tried to keep these instructions clear and correct, but the end result depends solely on you the builder.

Shavings and dust should be avoided. This is an old part, and no information is available concerning any hazards involved. Use a vacuum to remove dust and particles so you don't breathe a possible health risk.

You should work in a static free area the ACIA especially is easily damaged by static discharge, and the other parts are not immune. If you get a big spark when you walk across a carpet and touch a grounded object, wait for another day to work on anything electronic!

Test your DCM pak to be sure it works at high speed. This can s troubleshooting time later in case there is a problem with it. It's your choice whether to repair a bad DCM pak, or to just replace it. Very little of the existing circuitry will be used, so any problems

## How to reach The OS9 Underground:

For subscription information, questions, call or write to:

**Fat Cat Publications**
**4650 Cahuenga Blvd., Ste #7**
**Toluca Lake, CA 91602**
**(818) 761-4135 (voice)**
**(818) 365-0477 (fax)**
**(818) 769-1938 (BBS)**
or by email to:
**EditorOS9U@AOL.com**
**ZOGster@Delphi.com**

Article Submissions may be sent to the above addresses or to the following special internet account set up for that purpose:

**OS9Under@AOL.com**

Submissions should be sent as plain text and unformatted.
Join us on the New OS9 Underground Email, Mailing List:

**TheFatCat@AOL.com**

To Subscribe, just send email to the above account and you will be added to the list.

Change of Address:
If you anticipate a change of address, you must notify Fat Cat Publications at least 30 days prior your move to continue uninterupted service. We DO NOT re-send returned mailed. To re-claim returned mail, you must send $1.00 postage and handling per issue to the above address.

Back Issues:
Back issues are available at the cover price. Call or write for more info.


**Fat Cat Publications**

may be eliminated with the unused parts, but a known good unit will help. This would be a good time to download one or more terminal programs, since we will remove the ROM in the DCM.

Next, test your cables. Cable manufacturers are not perfect, and some cables that look right may be wired strangely. Make sure the correct pins are connected at each end, and that there are no shorts between pins. Repair, replace, or exchange as you prefer. The connections are shown in a table.

Now to the good part: Open the DCM case. Remove the black label on the bottom which will "void your warranty". Remove the screw under it. Release the snaps by pushing the bottom case half toward the switches and the top half toward the edge connector. The connector end will release so you can pull the halves apart. It's easiest to release the snap on one side at a time. If you have opened other packs this will be a familiar opera tion, if not just take your time and try to not break the case. Fold the case halves sideways. They will only fold one way because of a short grey cable connecting the main circ board to a small satellite LED board. Remove one screw and lift the small LED board out. Set the case top aside for now. Remove the 4 screws and lift the PCB (*Printed Circuit Board*) from the case bottom. Fold a strip of aluminum foil over the edge fingers for static protection. Hold it in place with a rubber band. Remember to remove the foil before you plug t board in for testing. You can get accustomed to seeing it there that you don't notice it, and it will short out everything. Use a wire wrapper/unwrapper to push out the 4 pins and remove the shield from the PCB. Set all the parts asi except the PCB.

Now remove the parts you no longer need. The fastest and easiest way is to just cut them off with a diagonal cutter and nail clippers. You may non-destructively re-







# Writing a Device Driver

## Part IV

by Boisy Pitre

This month we finish our analog/digital device driver by adding customized sound record and playback capability.

Recording and playback of sampled sound is a very specific application which doesn't readily fit within the specifications of our driver. However, OS-9 provides an easy means of interfacing specialized operations through the use of GetStat (Get Status) and SetStat (Set Status) calls.

Playback of sound is analogous to "setting" data into our device, while recording data from our device is similar to "getting." It does not matter which routine actually performs the recording or playback of sound; however, consistency's sake we will assign sound playback as a SetStat call and sound recording as a GetStat call.

Now that we have made provisions for recording and playback, we need to define a name and function code for each of the actions. Following Microware's naming convention for status calls, we will define playback as SS_Play, and recording as SS_Record.

We must also assign a unique function code to the respective name.

Codes 0-127 are reserved Microware's use, leaving us 128-65535.

I've chosen SS_Play as 151 and SS_Record as 150.

## TIMING ISSUES

We hit a major stumbling block when dealing with the issue of timing. *Why?* Because most sound files are recorded at ranges from 11KHz to 22KHz. This means that even at a low rate of 11KHz, one byte of data must be read from the A/D port every 1/11000's of a second.

This brings up some problems under a real-time operating system like OS-9.

OS-9's timing granularity, known as a tick, is normally 1/100's of a second. That is 110 times slower than what i necessary to create a sample of mediocre quality. Therefore, we cannot sleep even for one tick while waiting for the next byte to be read — it is simply too long a wait. Nor can we use a cyclic alarm since it granularity is in ticks as well.

Without DMA (as is the case with our example), our only solution is to loop inside the driver. It isn't the most desirable solution, since looping inside a driver keeps OS-9 locked in system state and use processes do not run until the driver' routine is exited. Howevei interrupts from other devices are still processed (typing on the keyboard while looping in the driver will save the keystrokes in the SCF buffer).

Due to this limitation (which should impress upon you the necessity to use interrupt driven devices), we are forced to compute a delay prior to assembling the driver. This delay value will provide timing base from which our driver wi derive the proper delay as close to the real rate as possible. Doing this in the driver prevents applications that use these functions from knowing the details of timing.

The delay value is dependent upon your computer's speed (in cycles per second) as well as the number of cycles takes to execute the looping code in the driver. Even more variables can come

```
/* crypt.c */
/*****************************************
*      COPYRIGHT NOTICE      *
*****************************************
* This software is
* copyright (C) 1993 by Bob van der Poel Software
* Permission is granted to reproduce and distribute
* this file by any means so long as no fee is charged
* above a nominal handling fee and so long as this
* notice is always included in the copies
* Other rights are reserved except as explicitly granted
* by written permission of the author.
*   Bob van der Poel Software
*   PO Box 355, Porthill, Id, USA, 83853
*   PO Box 57, Wynndel, BC, Canada, V0B 2N0
*   Phone 604-866-5772
*   CIS 76510,2203
*/

#include <stdio.h>
#include <modes.h>

#define BLOCKSIZE (10*1024) /* use 10K file chunks for speed */

extern int errno;

char buffer[BLOCKSIZE];

main(argc,argv)
int argc;
char **argv;
{
 register int i, count;
 int inpath,outpath;
 register char *key1ptr, *key2ptr, *buffptr, *keystart, *keyend;

 /* no filename, show help */

 if(argc!=3) usage();

 /* if any arg starts with -, help */

 for(i=1; i<argc; i++)
 {
  if(argv[i][0]=='-') usage();
 }

 /* open two paths to the file; one read, one write */

 if( (inpath=open(argv[1],S_IREAD))==-1 ||
   (outpath=open(argv[1],S_IWRITE))==-1 )
 {
  fprintf(stderr,"crypt: can't open %s\n",argv[i]);
  exit(errno);
 }

 /* set up pointers to the start/end of the key */
 key1ptr=keystart=argv[2];
 key2ptr=keyend=keystart+strlen(keystart);

 if(strlen(keystart)<3)
 {
   fprintf(stderr,"crypt: key must be at least 3 characters\n");
  exit(1);
 }

 /* process the file */

 while((i=count=read(inpath,buffer,BLOCKSIZE))>0)
 {
  for(buffptr=buffer; --count; )
  {
   if(*key1ptr=='\0')
   {
    key1ptr=keystart;
    key2ptr=keyend;
   }
   *buffptr  ^= *key1ptr++;
   *buffptr++ ^= *key2ptr--;
  }
  write(outpath,buffer,i);
 }
 close(inpath);
 close(outpath);
}

usage()
{
 register int t;

 static char *msg[]=
 {
  "",
  "Crypt, (c) 1993, Bob van der Poel Software",
  "Usage: crypt <filename> <key>",
  "Function: encrypts a file using a user supplied key",
  "",
  "***** Caution: Don't forget the key--not even god *****",
  "***** can decrypt the file without the exact key! *****",
  ""
 };

 for(t=0; t<sizeof(msg)/sizeof(msg[0]); t++)
 {
  fputs(msg[t],stderr);
  putc('\n',stderr);
 }

 exit(0);
}
```

(EOF)





move them for a later project if you wish, but remember that the most important part for this project is the PCB, so don't damage it. No removed parts will be reused for this project.

Remove: The ribbon cable and the LED board, SW1, SW2, JK1, SA1, T1, D1, D2, VR1, R1, R2, R3, R4, R5, R6, R8, R16, C4, C5, C6, C7, C8, C9, C10, C15, C20, IC1, IC5, X2. That is, everything from the large notch to the switch end of the PCB, plus IC1. There is no magic order to this. Just cut whatever you can get to most easily. Leave: IC2, IC3, IC4, X1, C1, C2, C3, C11, C12, the unmarked capacitor near R1, and the diode on the bottom of the PCB.

Next, remove some unneeded traces. Using a sharp knife, cut a section of copper from both sides of IC2 pin 17 on the bottom of the PCB: make a cut near the pin and another about 1/8 inch away. Work the knife under the copper and peel it up. Be careful to not let the knife slip and cut any of the other traces. Now, cut the trace at pin 16 and follow it to the pad near pin 14, cut it there, and peel it off. Repeat this with the trace at pin 9 on the top. Find 3 short traces, 2 narrow and one wider, between IC2 pins 1 and 28, going to solder pads. They come from pins 12, 9, and 10. Find the pads on the bottom and peel off the traces there. Also remove the wide trace running between them on the bottom of the PCB, running nearly the full width of the PCB, to make room to drill a hole later. File or sandpaper away any rough places in the newly bare areas, remove any solder bridges and generally clean up the PCB. Use paint thinner or alcohol to remove any flux. Lightly sand the green paint on the top to let the glue adhere later. Just roughen it, don't sand through to the copper.

Modify the 9 pin RS232 connectors. Some connectors have pins that extend through the plastic base to be soldered into a PCB. Clip the pin ends that would normally extend through a PCB close to the plastic body of the connectors. Place the connectors on a flat surface and make sure the leads don't reach that surface. Mix some epoxy and spread it over the cutoff pins and the plastic body where it is recessed slightly. Lay it, epoxy side-up, on a plastic wrapped covered flat surface to set. Clamp or weight it down enough to keep it in place. Alternatively, use instant glue to hold a thin (.020) strip of plastic to the bottom of the connector body. Trim and drill the plastic strip to match the connector base after the glue sets. Some connectors don't have the pins run through the base. They are unsupported at the PCB end. Just bend the pins out straight and wire to them. You may clip them shorter to save space. The purpose of all this is to keep the pins from contacting any traces on the PCB and shorting out the signals, or letting the magic smoke out of an IC, while allowing easy connections to the pins.

Now modify the PCB. Since it isn't wired for +/- 12V, you must add 2 wires for that. -12V is easy, since pin 1 is on top. +12V is more dificult; pin 2 is on the bottom. A hole allows a wire to pass from the bottom to the top. Holes will be needed to mount the new RS232 connectors. When you hold the PCB up to a bright light, you can see through where there is no copper on either side. Locate a clear area near pin 1, mark it and carefully drill a hole large enough to clear the insulation on a 30 gauge wire. Don't drill through any traces! An alternate method is to use a thin saw (like a coping saw) or a file to make a notch near pin 1 at the base of the large slot. Cut just deep enough to clear

a 30 gauge wire, and avoid the copper ground planes. Later, you can glue the wire in place. Place the PCB in the case bottom. Mount the 9 pin connectors on the cables. Lay the connectors in place on the end of the PCB to check the fit. You probably will need to widen the opening in the case slightly.

Use the nail clippers, diagonal cutters, and file for this. Save the latching tabs. If you are building only one port now, leave room for a second connector, or even mount two now to make adding a second port easier. The connectors can extend from the case, but keep the mounting holes far enough on the PCB for strength. Remember those connectors must support the cables. When the fit is acceptable, mark the PCB and drill clearance holes for your mounting screws. Oversized holes in the PCB let you move the connectors slightly, lining them up for a neater appearance. Peel away any traces the screws may touch, excepting ground. Separate the cables and connectors. Use a larger drill bit to deburr all holes, so you won't nick the insulation when you run wires through them. Be careful not to remove too much material. Mount the connector retainers (the "long nuts") then mount the connectors on the PCB. Trim the case halves to fit around them, bottom first, then the top. Finish the case now and take time to make it look good - this is the part everyone else will see. Maybe you can get help from a plastic model builder you know.

Mark the 1488 and 1489 ICs on the underside with their part number and pin 1 location. You can't read the tops after they're glued down! Use epoxy or instant glue to mount them with pins up between IC1/IC2 and the 9 pin connectors. Put them in line, pin 1 of each toward the 9 pin connector, with the 1489 nearest the ACIA. This keeps all the parts for one port in a row so they're easier to wire. Leave a little room between rows (ports) for placing wires - 1/4 inch is enough. There will be lots of space for expansion in case you want to add something else to this pack.

You can now drill another hole near the end of IC2 to run wires through, you can just run wires around the edge, or you can connect to IC2's pins on top of the PCB.

If you have a favorite method wiring point to point, by all means use it. Leave about 1/4 inch of slack in each wire so you can move it out of the way for the later wires. Where I can, I wrap the wires onto the pins for a better connectic Strip about 1/4 inch and wrap as on wire-wrap pin. Strip both ends before connecting the first end. These pins are meant to be soldered, not wrapped, so be gentle. To connect to pins on IC2 and IC4, remove most of the solder, strip 1/8 inch of insulation, bend a hook in the bare end, squeeze it onto the pin, and resolder. Don't leave any bare wire to cause shorts. Wire to the 9 pin connectors the same way. Stagger the level of the wires between pins for more room. Wire to the inner/lower pins first so you can rea between the others to solder them without burning the wires on the outer/upper pins.

Run the new wires. Try to use unique colors for the connections between different parts. A widely used standard is to use red for +5V and black for ground. You may use only one color for all connections; electrons don't care, but that will make troubleshooting more difficult.

I don't recommend using a Y-cable. If you must, modify the following +/-12V and +5V wiring to use a separate power source, either a voltage converter, a wall plug, or a power supply. You can install a polarized plug and socket with at least 4 pins for +12, -12, +5, and ground. Connect your ground lead to any grou point, and use the following to wire the other voltages to the ICs. Use the pinout

# Tales from the Crypt

By Bob van der Poel

I've got a teenage daughter who is now using my computer...however, she is a bit hesitant about having all her "private stuff" on my hard drive. And she already knows that the password protected login system might keep her brother from reading her files...but not me. So I explained how one can encrypt a file so that no one can read it. She figured that was a great idea, and I went looking for the program. Of course, I couldn't find it, (*well, I did find a couple, but the were inadequate for various reasons*). After writing this, I figured that with communications being the theme of this issue, what better topic than the prevention of certain communications, (*especially where teenage children are concerned*).

There are many different methods of encrypting a file. As a matter of fact, entire books have been written on the subject. The code presented here can probably be broken by the CIA, but I don't think they care too much about my daughter's "private stuff".

Crypt uses a simple method to make the original message unreadable. It takes each character in the original message and EORs it with a character in a user supplied "key". To return the message to its original state, the characters are EORed a second time with the same key.

To make things a bit more secure, my version of the program does the EOR

twice for each character. This prevents a partial key from producing a partial decoding. The following program will work on OS9/68000 and OS9/6809 machines...and the files can be encoded on one system and decoded on the other. You can even use crypt to encode binary programs if you want (for example, those x-rated gifs you might have).

But a word of caution: The degree of security you achieve is directly proportional to the length of the key—the longer the key, the more difficult it is to decode the message; however, long keys may be harder to remember!

And if you forget the key you will not be able to decode the message. Also, the result of the encryption will not be transmittable as text on a BBS or other electronic service—you will have to transmit the file as "binary".

To use crypt, just pass the name of the file followed by the key as parameters. For example:

```
crypt diary secret
```

will encrypt the file "diary" using the key "secret". To return the file to its original format, just type the same command.

By using large block reads and over writing the existing file in place, cry works quite fast. I timed it on a 230K file on my MM/1 and it took 4 seconds to process the file.

If you'd like to comment on this program, or any other OS9 topics, drop me a note. I can be reached at:
PO Box 355, Porthill, ID 83853 or
PO Box 57, Wynndel, BC, Canada V0B 2N0 or
Compuserve 76510,2203.





of your power source to wire to it. If you provide a separate 5V source, cut the trace between C1 and the edge connector pin 9. Test your power supply before you connect it to your DCM Pak.

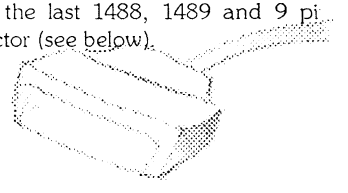
Wire +/-12 volts. The edge connector pin 1 is -12V, pin 2 is +12V. Pin 1 is marked on top, pin 2 is at the same end on the bottom. Cut wires long enough to reach from those pins along the side past X1 to the 1488 ICs, plus some extra. I use a felt tip marker to put stripes on these wires to help identify them later. Strip 1/16 inch, hold the bare end across the PCB end of the pin and solder. Use very little solder so it won't run up the pin. Don't touch the wire insulation until it cools or you will crush it and expose bare wire. Run the wire from pin 2 through the hole or slot you made. Connect the other ends, +12V to pin 14 of both 1488s, -12V to pin 1 of both 1488s. Leave plenty of slack so you can route these wires well away from the rest of the circuitry. Glue them to the PCB so they can't slip under a mounting screw.

Wire +5 volts. Use your ohmmeter or continuity tester to find the +5V pads; they are all over the board. Look for continuity to the side of C1 that connects to the edge connector pin 9. You will need to temporarily remove the aluminum foil from the edge connector - it connects all the pins together. Use pads that are close to the 1489s pins 14. The 1488s don't use +5V. Mark the pads to use, replace the aluminum foil, then wire.

Wire the ground connections. There are even more potential ground connection points than there were for +5V. Pick pads that are in narrow copper areas; they are much easier to solder. Connect a ground to the 9 pin connectors, the 1488s pins 7 and the 1489s pins 7. You may need a soldering iron with higher wattage to make good connections to ground.

Wire IC2, one 1488, one 1489, and 9 pin connector (see tables). It is good idea to test this port now, while the ACIA is easy to replace. You will need a T3 descriptor - see below.

Bend up pins 2, 5 through 12, 16, and 17 of the new ACIA. Make them stick straight out from the case. Place the new ACIA on the old one, with both pins touching. Use 1 or 2 strips of thin cardboard between them as a spacer to allow air circulation. Bend the pins of the new ACIA as necessary until it sits on the old ACIA with all the pins touching. Recheck the orientation, then solder 2 corner pins (1 and 15). Check again, adjust if needed, then solder all touching pins together. Use a minimum of heat and solder, and take some time between pins to let things cool. Slip the cardboard strips out. Now wire the straightened pins of the new ACIA, the last 1488, 1489 and 9 pin connector (see below).



Wire the new crystal to the new ACIA. Trim the crystal leads short, wrap a 1 inch length of stripped 30 gauge wire on each lead and solder. Don't overheat. Insulate both leads. Wrap the other end of th wires to pins 6 and 7 of the new ACIA Position the crystal on and fasten it to X1. Double sided tape or glue will keep it in place.

On the bottom of the PCB connect a wire from the old IC1 hole 8 to IC4 pin 1 and connect IC4 pin 2 to your new ACIA pin 2. If you are building only one port, you can change the address from $FF6C to $FF68, the same as the Deluxe RS232 Pak. Most if not all DECB program expect the ACIA at $FF68, and you can use the stock T2 under OS-9. Cut a 1/16 inch section from the trace going to IC2 pin 2, wire IC2 pin 2 to IC4 pin 2, and wire

```
(* convert dentry into a name *)
name:=""
IF dentry(1)>0 THEN
 FOR i:=1 TO 29
 EXITIF dentry(i)>127 THEN
  REM last character in filename
  name:=name+CHR$(dentry(i)-128)
 ENDEXIT
  name:=name+CHR$(dentry(i))
 NEXT i
ENDIF

 (* now we add it to pathlist in reverse order *)
 pathlist:="/"+name+pathlist
ENDLOOP

(* comes here when we got root dir.
(* get device name

regs.a:=path
regs.b:=$0E
regs.x:=ADDR(dentry)
RUN syscall($8D,regs)
CLOSE #path

name:=""
FOR i:=1 TO 32
 j:=dentry(i)
EXITIF j>127 THEN
 name:=name+CHR$(j-128)
ENDEXIT
 name:=name+CHR$(j)
NEXT i

pathlist:="/"+name+pathlist

(* if subroutine must restore current
(* working directory
CHD pathlist
PRINT pathlist
END
```


-Mike Guzzi

# HANGUP

BY: WAYNE CAMPBELL

This months theme is communications, so I thought I'd take the opportunity to discuss the problem of illegal logoffs when a Basic09 subroutine module is in use.

As those of you who run (or have run) the StG Login Package (and used Basic09 subroutines to handle various user tasks, such as the original NR (News Reader) know, if a user logs off illegally (and a Basic09 subroutine is the current process), the I-Code modules are left hanging in memory and mus be manually unlinked to kill them. You also know that trapping the process abort error (228) does nothing.

Well, I am now going to explain why this happens. Some of you may already know this, but, for those of us who are less enlightened, the information presented here may be of some use.

When a process is aborted, it means that OS9 received a signal. With user-defined signals, the signal must be intercepted by the process to avoid abortion. However, when a user logs off illegally, the signal sent is the kill process signal (signal 0), which is non-interceptable (almost). The exception is, when a process is in system state, the kernel prevents the abortion from taking place until the process returns to user state.

Now, the reason that the I-Code subroutines are left hanging in memory is because the only module being unlinked in a process abort is the PRIMARY module (in this case, RunB). The subroutines just stopped execution, and weren't even aware that an error occurred. And, since at least one of the subroutines in use was linked to the process (and was NOT unlinked), the module(s) were left in memory, unable to be killed.

I think that there may be a way to 'fix' this problem, though someone with more knowledge of the kernel would know more. It would involve a patch to (or re-write of) the kernel that would cause the system to suspend the process abort, check to see if the process' primary module is RunB, and then proceed. If the process' primary module is RunB, then the kernel would pass the process abort error code (228) to the process, wait for the process to abort the subroutines, and then proceed with the process abort in the usual manner.

Since RunB leaves it to its subroutines to check for errors, the subroutines would have to trap the 228 error code, or the procedure would just error out (and possibly get hung in an error loop if there was error trapping that didn't account for the 228 error).

If someone reading this (who can answer the question of patching the kernel) wishes to respond to that question, you can write to me, care of this magazine.

-Wayne Campbell

# BASIC Training by Jim Vestal

This month I present a Basic09 program submitted by Mike Guzzi, called BPWD, (*thanks Mike!*). It will return the path name of the current working directory, from within a Basic09 program. He modeled his program after the OS-9 command PWD.

Interesting to note that Wayne Campbell also submitted a similar program called PD and wrote an article explaining the theory behind finding the current directories (both the current working and execution directories). See his article for all the technical details.

Both BPWD and PD use similar methods to accomplish the same thing.

I enjoyed comparing the individual programming styles of both Mike and Wayne.

I hope that you too will take some time and look over both programs then decide which one you will use within your own projects.

Till next time,

-Jim Vestal

```
PROCEDURE bpwd
   (* pwd command in basic09       *)
   (* taken from dissassembly of pwd *)
   (* by Mike Guzzi for use of the OS-9 Underground *)
   TYPE registers=cc,a,b,dp:BYTE; x,y,u:INTEGER
   DIM pathlist:STRING[128]
   DIM name:STRING[32]
   DIM dentry(32):BYTE
   DIM path:BYTE
   DIM i,j,k:INTEGER
   DIM regs:registers
   DIM p_lsn(3):BYTE
   DIM c_lsn(3):BYTE
   DIM t_lsn(3):BYTE
   (* start of code *)
   pathlist:=""
   (* open current dir. get current LSN and parent LSN *)
   OPEN #path,".":READ+DIR
   GET #path,dentry
   p_lsn(1):=dentry(30)
   p_lsn(2):=dentry(31)
   p_lsn(3):=dentry(32)
   GET #path,dentry
   c_lsn(1):=dentry(30)
   c_lsn(2):=dentry(31)
   c_lsn(3):=dentry(32)
   (* this is the main loop
   (* compare LSN's if the same get device
   (* name and exit *)
   LOOP
   EXITIF p_lsn(1)=c_lsn(1) AND
      p_lsn(2)=c_lsn(2) AND
      p_lsn(3) =c_lsn(3) THEN
   (* found root dir
   ENDEXIT
     CLOSE #path
     CHD ".."
     (* now we must get the old "." entry name open
     (* parent and get new LSN's then find the name
     t_lsn:=c_lsn
     OPEN #path,".":READ+DIR
     GET #path,dentry
     p_lsn(1):=dentry(30)
     p_lsn(2):=dentry(31)
     p_lsn(3):=dentry(32)
     GET #path,dentry
     c_lsn(1):=dentry(30)
     c_lsn(2):=dentry(31)
     c_lsn(3):=dentry(32)
     LOOP
     EXITIF EOF(#path) THEN
       REM no more entrys
       PRINT "Error - Can't find dir entry"
       CLOSE #path
       END
     ENDEXIT
       GET #path,dentry
     EXITIF dentry(30)=t_lsn(1) AND
          dentry(31)=t_lsn(2) AND dentry
          (32)=t_lsn(3) THEN
       REM found dir entry
     ENDEXIT
   ENDLOOP
```





IC4 pin 1 to IC1 hole 8. You can move the wire at IC2 pin 2 and restore the original connection later, if you add a second port, by replacing the cutout section with a short wire.

Use the following table to wire the ACIAs to the 1488s and 1489s. Pins 9, 10, and 12 of IC2 are available at the solder pads between pins 1 and 28 where you removed traces on the bottom. Run the wires from IC2's other pins through the hole you drilled, or around the edge. Both ends of all the wires from the new ACIA will be on top of the PCB.

| ACIA pin | 1488 pin | 1489 pin | Name |
|---|---|---|---|
| 8 | 4+5 | - | RTS |
| 9 | - | 3 | CTS |
| 10 | 2 | - | TXD |
| 11 | 9+10 | - | DTR |
| 12 | - | 6 | RXD |
| 16 | - | 8 | DCD |
| 17 | - | 11 | DSR |
| 5 | n/c (no connection) | | |

Above, 4+5 and 9+10 means connect those pins together and to the ACIA. Each ACIA connects to only one 1488 and one 1489. Go through the table for each ACIA.

Use this table to connect the 9 pin connectors to the 1488s and 1489s, and to check your cable wiring.

| 25 pin | 9 pin | 1488 pin | 1489 pin | Name |
|---|---|---|---|---|
| 22 | 9 | n/c (no connection) | | |
| 5 | 8 | - | 1 | CTS |
| 4 | 7 | 6 | - | RTS |
| 6 | 6 | - | 13 | DSR |
| 7 | 5 | Wire to any ground | | GND |
| 20 | 4 | 8 | - | DTR |
| 2 | 3 | 3 | - | TXD |
| 3 | 2 | - | 4 | RXD |
| 8 | 1 | - | 10 | DCD |

The table has the 9 pin connector pin numbers in reverse order to remind me to remind you to wire the lower row first.

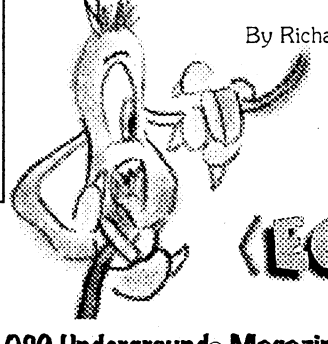
Again, each 9 pin connector is wired to only one 1488 and one 1489. Go through the table for each connector.

Check your work. Look for wir clippings, solder splashes, damaged insulation, shorts, poor connections, etc. Fix it now. You can hold the wires to the PCB with silicone glue/caulk. Solder all connections to make them permanent. At tach the shield with the 4 pins. You may need to trim it to clear the 9 pin connector mounting screws. Mount the PCB in the case with the four screws, remove the foil, and clean the 40 pin edge connector Mark the 9 pin connectors as T2 and T3; T2 is connected to the new ACIA. Snap the case top on.

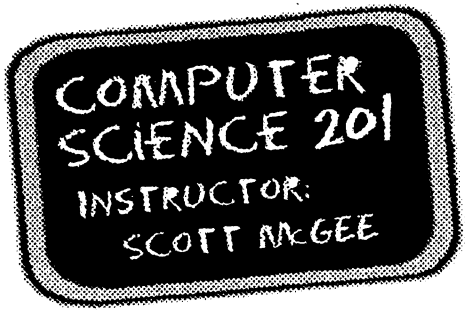
The T2 descriptor can be used as is, and a copy can be patched to provide a T3 descriptor. Since both ports will be in the same Multi-pak slot, you can't use th stock T3; it's for slot 2. Dump your T2 module. Look for two bytes with "FF68" and two bytes with "54B2". In my stock T2 they are at offset 000F+0010 an 002E+002F respectively. Patch them to be "FF6C" and "54B3", update the CRC with your preferred utility, and save as T3. Build a new OS9boot including T2 and T3.

Plug your new 2-port RS232 pack into slot 1 of your Multi-Pak. Without the aluminum foil! Now show your pride in your new creation: Cover that "DCM label with a new one proclaiming: "*<Your name here>'s Dual RS232 Pack*".

By Richard Albers



(EOF)




# Lesson 3: Doubly Linked Lists

Instructor's Note:

Before I start today's lesson, let me say a word about homework. I realize that there are a wide variety of readers who will read this column. Each article contains a homework assignment. *I do not* expect everyone to actually code the assignment. That would be kind of silly if the reader was a long time programmer, familiar with the material, and just reading the article to see if I have something new or interesting to say.

On the other hand, if you are new to these concepts, I highly recommend that you *do* try to code the homework assignments. I also expect that some people may have problems with some of them. If you happen to fall into this category, please feel free to contact me by any of the means presented in each article, (email, US Mail), for help with your code.

I would love to have someone who is struggling with a new concept learn it because I was able to help. I won't be issuing any grades but if you are learning new concepts here, please do the homework. And please, ask me if you have any trouble with it.

Errata:

Last time, I made a typographical error in my discussion. At one point,I said: **"if(list = NULL)"**

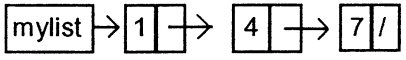
Unfortunately, this will ASSIGN a NULL to list!

It should have read:

**"if(list == NULL)"**

I hope this didn't inconvenience anyone. *(oh, and an extra 10 points if you caught it!)*

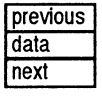
Previously, we have discussed Linked Lists. Today I wish to discuss a common variation of the linked list called the Doubly Linked List. A standard linked list looks like t following:



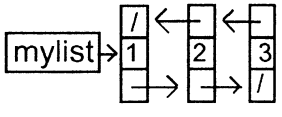

With a doubly linked list, in addition to the pointer to the next node, we have a pointer to the previous node. Thus, the declaration for it may look like:

```
typedef struct DLLIST *dllist; struct
DLLIST{ int data; dllist next; dllist
        previous; }
```

Graphically, each node would look like this:




and a list would look like:




Here, "mylist" points to the first node, which contains one, (1). The "previous" pointer in that node is NULL, (indicated by the '/'), to show that it is the head of the list and that there is no previous node. In the last node, the "next" pointer is NULL, indicating the end of the list, while the "previous" points back up the list. You will recall the parent pointers we had to use with a standard





```
(*
(* If not the root dir
IF Parent_Sector<>Child_Sector THEN
 REPEAT
  (*
  (* Open next dir
  IF Data_Dir THEN
   CHD ".."
   OPEN #Dir_Path,".":READ+DIR
  ELSE
   CHX ".."
   OPEN #Dir_Path,".":READ+EXEC+DIR
  ENDIF
  (*
  (* Get '..' and '.' from current dir
  GET #Dir_Path,File_Entry
  (*
  (* Determine sector numbers
  Byte_Sector:=File_Entry(1).Sector_Byte_ID
  Int_Sector:=File_Entry(1).Sector_Int_ID
Parent_Sector:=Byte_Sector*65536.+Int_Sector
  (*
  Byte_Sector:=File_Entry(2).Sector_Byte_ID
  Int_Sector:=File_Entry(2).Sector_Int_ID
 Child_Sector:=Byte_Sector*65536.+Int_Sector
  (*
  (* If '..' and '.' are the same then current
  (* dir is root dir
  IF Parent_Sector=Child_Sector THEN
   Is_Device:=TRUE
  ENDIF
  (*
  (* Get the entry for the previous dir
  Child_Sector:=Temp_Sector
  REPEAT
   GET #Dir_Path,Check_Entry
   (*
   (* Determine sector number
   Byte_Sector:=Check_Entry.Sector_Byte_ID
   Int_Sector:=Check_Entry.Sector_Int_ID
Check_Sector:=Byte_Sector*65536.+Int_Sector
   (*
  UNTIL Check_Sector=Child_Sector
  CLOSE #Dir_Path
  (*
  (* Set '.'
  Last_Entry:=File_Entry(2)
  (*
  (* Determine sector number
  Byte_Sector:=Last_Entry.Sector_Byte_ID
  Int_Sector:=Last_Entry.Sector_Int_ID
 Temp_Sector:=Byte_Sector*65536.+Int_Sector
  (*
  (* Get the name of the previous dir
  Pointer:=0
  Current_Name:=""
   REPEAT
    Pointer:=Pointer+1
Current_Name=Current_Name+CHR$(LAND(Check_Entry.Name
      (Pointer),127))
   UNTIL Check_Entry.Name(Pointer)>127
   (*
   (* Add name to pathlist
   Path_List:="/"+Current_Name+Path_List
   (*
   (* If the root dir then add the device name
   IF Is_Device THEN
    Path_List:=Device_Name+Path_List
   ENDIF
  UNTIL Is_Device
  (*
  (* Now chx or chd back to original dir
  IF Data_Dir THEN
   CHD Path_List
  ELSE
   CHX Path_List
  ENDIF
 ELSE
  (*
  (* If the root dir then set pathlist to device name
  Path_List:=Device_Name
 ENDIF
 (*
  (* Print pathlist; To return Path_List to a calling
procedure
 (* without printing, delete or REMark out this line
 PRINT Path_List
 (*
 (* End of program
 END
```

-Wayne Campbell

Email: "zog!wayne@abode.ttank.com"
or "os9under@AOL.com"

(EOF)

linked list? Well, with a Doubly Linked List, this is no longer needed.

To do a insert into a doubly linked list, we do the following:

*(Please note that I will now begin using pseudo code, rather than 'C' code )*

```
if node = NULL list = new else
if node->data > target new->next = list
list->previous = new list = new else
node = list while node->data < target AND
node->next != NULL
node = node->next new->previous = node
new->next = node->next  node->next = new
new->next->previous = new
```

Notice that this method handles an empty list by simply assigning the list to point at the new node. It handles insertion at the beginning of the list in a similar manner, but sets "**new->next and list->previous**" first. If the new node belongs somewhere after the first node, we enter the third section of the "**if**" clause. Here, we start by initializing node with list. After the while loop, node points at the last list item who's data is *less* than target. The next line lets new's previous pointer point back at this node. The following line sets new's next pointer to point to the next node in the list. At that point, "**new**" thinks it is in the list, but none of the list nodes know about "**new**". To fix this, we have to set the next pointer on node and the previous pointer on the next node. One thing to remember here is that, in the next to last line, we have just redirected node's next pointer to point to new. If we then set **node->next->previous**, we would be changing new's pre' pointer. Since new->next now points to the next node, we need to use it instead.

## Ok, now lets take a crack at deletion.

Again, I will do it in pseudo code. If you are new to pseudo code, remember, it is just a way to explain the process without getting caught up in the syntax. My pseudo code tends to look a lot like 'C' because I normally program in 'C'. I have seen other pseudo code that was straight English text with no symbols. The idea is to convey the meaning, not to be able to compile it!

Lets look at the following:

```
if list = NULL
      quit
if list->data = target list = list->next
      free list->previous
if list != NULL list->previous = NULL
else node = list
while node->data < target AND node->next != NULL
      node = node->next
      if node->data != target
            node = NULL
      if node != NULL
            node->previous->next = node->next
if node->next != NULL
      node->next->previous = node->previous
free node
```

This code looks more complicated than it is. Lets take it a bit at a time. First, we check for an empty list. We'll just bail out if we have one.

```
if list = NULL, quit
```

Next we have to handle the case of the node to delete being the first node. This is a special case since we have modify the pointer to the list. The first thing we do is to step the list pointer past the node. We can do this without having a pointer to the node because we have the previous pointer to use. Using that, we then free the node.

```
if list->data = target list = list->next free list->previous
```

Now, we must update the previous pointer to show that the node is now the head of the list by setting it to NULL.

```
if list != NULL list->previous = NULL else node = list
```

Finally, if the node is not at the head





```
IF Exec_Dir<>"-X" AND Exec_Dir<>"-x" THEN
   ErrorMsg:=" pd: unknown option"
   END ErrorMsg
ENDIF

(* If parameter then check execution dir
   Data_Dir:=FALSE
   (*
   (* Go here if no parameter
1   er:=ERR
   ON ERROR
   (*
   (* Assume not root dir
   Is_Device:=FALSE
   Path_List:=""
   (*
   (* Open initial dir
   IF Data_Dir THEN
     OPEN #Dir_Path,".":READ+DIR
   ELSE
     OPEN #Dir_Path,".":READ+EXEC+DIR
   ENDIF
   (*
   (* Get device name for current drive
   Call_Code:=$8D
   Regs.A:=Dir_Path
   Regs.B:=$0E
   Regs.X:=ADDR(Device_Buffer)
   RUN SysCall(Call_Code,Regs)
   (*
   (* Set device name
   Device_Name:="/"
   Pointer:=0
   REPEAT
     Pointer:=Pointer+1
Device_Name:=Device_Name+CHR$(LAND(Device_Buffer(Pointer)
         ,127))
   UNTIL Device_Buffer(Pointer)>127
   (*
   (* Get '..' and '.' from current dir
   GET #Dir_Path,File_Entry
   CLOSE #Dir_Path
   (*
   (* Set '..'
   Check_Entry:=File_Entry(1)
   (*
   (* Set '.'
   Last_Entry:=File_Entry(2)
   (*
   (* Determine sector numbers
   Byte_Sector:=Check_Entry.Sector_Byte_ID
   Int_Sector:=Check_Entry.Sector_Int_ID
   Parent_Sector:=Byte_Sector*65536.+Int_Sector
   (*
   Byte_Sector:=Last_Entry.Sector_Byte_ID
   Int_Sector:=Last_Entry.Sector_Int_ID
   Child_Sector:=Byte_Sector*65536.+Int_Sector
   Temp_Sector:=Child_Sector
```

the same, PD assigns the Device Name to the Path_List String, prints the string, and exits.

If it is *not*, PD begins a REPEAT loop that CHDs (*or CHXs*) back one directory at a time until it reaches the root directory. With each pass, PD keeps track of the previous directory so that it can find the filename for that directory when it retrieves the parent and child entries of the current directory. It then adds that filename to the Path_List string.

Once it reaches the root directory, it assigns the Device Name to the Path_List string, prints the string, CHDs (*or CHXs*) back to the starting directory, and exits.

If you were just going to use it as a means of remembering which data or execution directory you are in from the shell, it is useless. As soon as the process ends, the paths established in the shell's process are reinstated.

The routine for CHDing or CHXing back to the starting directory *does* serve a useful purpose.

But, what if you wanted to use PD in a program by calling it? If that little routine wasn't there, you would find yourself in the root directory! The reason is, Basic09 (*or RunB*) is the process, *not* your program. So, Basic09 (*or RunB*) is going to assume that the last CHD (*or CHX*) done is the directory you want to be in. PD would return a string stating that your current data path, (*for example*), is **/D1/TEST/WIDGIT**, if your program was to do a **'SHELL "dir"'** at that point, you would find yourself in the root directory of /D1. In order to avoid this, PD *must* CHD (*or CHX*) back to its starting directory.

The above reference to calling PD from another procedure requires a minor change to the code:

Change the statement:

**DIM Path_List:STRING[255]** To: **PARAM Path_List:STRING[255]**

If you don't want PD to print the result, delete the statement: **PRINT Path_List**

Use the following syntax for calling PD from within your programs:

For Current Data Directory:

**RUN PD(<PathList>)**

For Current Execution Directory:

**RUN PD(<PathList>,"-x") or**
**RUN PD(<PathList>,"-X")**

**<PathList>** is a string variable of up to 255 characters.

## Finally:

If anyone can figure out the exact system call required (and the exact code to implement it) for OSK, send it to me, care of this magazine, or send me e-mail with all of the pertinent information.

Enjoy!

```
PROCEDURE pd

(* pd - print current data/execution directory path

(*.     Copyright (c) 1993 Wayne Campbell
(* usage:
(* pd              - print current data directory path
(* pd ("-x [or -X]") or
(* pd -x [or -X]    - print current execution directory path

(* Increase the length of Path_List if you run into problems *)

TYPE REG=CC,A,B,DP:BYTE; X,Y,U:INTEGER
DIM Regs:REG
TYPE ENT=Name(29),Sector_Byte_ID:BYTE;
   Sector_Int_ID:INTEGER
DIM File_Entry(2),Check_Entry,Last_Entry:ENT
DIM Device_Buffer(32),Dir_Path,Call_Code,
   Byte_Sector:BYTE
DIM Pointer,er,Int_Sector:INTEGER
DIM Parent_Sector,Child_Sector,Check_Sector,
   Temp_Sector:REAL
DIM Is_Device,Data_Dir:BOOLEAN
DIM ErrorMsg:STRING
DIM Device_Name:STRING[33]
DIM Current_Name:STRING[29]

(*To return Path_List to a calling procedure, change DIM to PARAM
DIM Path_List:STRING[255]
PARAM Exec_Dir:STRING[2]

(* Assume check current data dir
Data_Dir:=TRUE

(* Check for parameter
ON ERROR GOTO 1
```





of the list, we point a temporary pointer at the head, and then use the while loop to step down the list until we either find the location of the node to delete, or reach the end of the list. The while loop will end when the we reach the end of the list because node->next is NULL, or when node->data is no longer less than target.

```
while node->data < target AND node->next != NULL
      node = node->next
      if node->data != target
            node = NULL
      if node != NULL
            node->previous->next = node->next
```

Since node->data may or may not be equal to target, we use an additional test to set node to NULL if node->data is greater than target because we only got here if the last node was less than target, so if this one is greater, we know that target is NOT in the list.

At this point, if node is NULL, we know that target was NOT in the list. If not, then node is indeed the one we want. First thing we do from here is to set the next pointer on node->previous to the value of node->next to point it around node. Then we have to see if there is a node a node->next. If so, we need to set it's previous pointer to point back at node->previous.

```
if node->next != NULL
      node->next->previous = node->previous
```

Finally, now that we have isolate node, we must free it.

```
free node
```

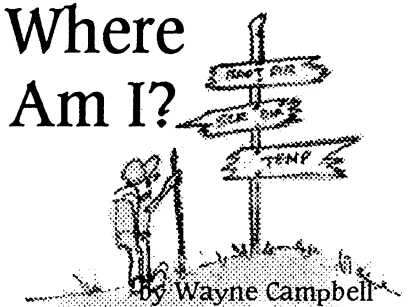
For your homework assignment, rewrite the pseudo-code presented her in your favorite programming languag and add the needed code to create and initialize new nodes.

Next time, we will use a very similar type definition for a very different kind of data structure... trees!

## Class dismissed!

(EOF) -Scott McGee

# Where Am I?




by Wayne Campbell

After I became familiar with the directory structure of OS-9, I found myself wanting to know how the commands PWD and PXD worked. Only now, after 4 years and upgrading to a CoCo3 and Level 2, have I learned how they work. This is especially important to me as a programmer, because there are times I would like to display the current data, (and/or execution), directory path in my Basic09 programs, but previously had to use, (in my opinion), inefficient code to do it.

## Note to OSK users:

**PWD** and **PXD** use a system call to retrieve the current device name. OSK users will have to figure out how to do the appropriate system call to get the device name.

## PD:

This procedure is a Basic09 PWD/PXD utility all in one program. To use it:

**PD** with no options returns the Current Data Directory Path. PD with the option "-x" (or "-X") returns the Current Execution Directory Path. PD with any other option returns the error message: *"pd: unknown option"*

PD was developed on a Tandy Color Computer 3 under OS-9 Level 2. It *SHOULD* run on any OS9 Level 1 or Level 2 platform.

## How it works:

First, the Specified, (Current Data or Execution), Directory is opened for read, and the parent, (..), and child, (.), entries are read. The system call to retrieve the Current Device Name is performed, and the current directory is closed.

The system call is a GetStat ($8D) call: (*Excerpt from the Level 2 manual*)

```
SS.DevNm (Function Code $0E)
         Returns a Device Name
Entry Conditions:
A = Path Number (Path to currently opened file)
B = $0E X = Address of 32-byte buffer for name
Exit Conditions:
X = Address of buffer, name moved to buffer
```

I suggest NOT using a 32-character string as the buffer (though it is leg: because Basic09 terminates any string using less than its maximum allocation with $FF as the first character after the last character of the string. Example:

```
name$:="DD" (name$ is 32 characters in length)
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2
-------------------------------
D D .(the rest is garbage) ----------------
4444FF
(the rest is garbage)
```

This is what name$ would contain if used to receive the results of the call.

```
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2
-------------------------------
44C4(the rest is garbage) ----------------
D D
(the rest is garbage)
```

As you can see, the Device Name would contain garbage, as there's no end-of-string marker. This is why PD uses a 32-byte array to get the name, and then build the string. Notice that SS.DevNm does NOT include a leading slash '/'. This must be added to the string.

Next, PD determines if the current directory is the root directory. In the root directory, '..' and '.', have the same LSN. If it IS, PD assigns the Device Name to the Path_List string, prints the string, a exits.

If the parent and child directories are