SUPERPET GAZETTE

This issue is our 8th; next issue marks the end of our first year; September is the anniversary of the founding of SPUG. Our name has now changed; because of the large number of members in Canada and Europe we are now the International SuperPET Users Group, or ISPUG, thanks to the members and their strong support with dues, comments, and articles. We hope our 2nd year is as successful as the first. Which leads us to the subject of:

****

DUES SOON DUE! ARE YOU ABOUT TO EXPIRE? SEPTEMBER MEMBER, YEAR'S ABOUT UP!

On our original publication schedule, we promised 8 issues. Our first was a flyer to get organized, so next issue will be the last for those who entered ISPUG in September '82. Check the address label on this issue. If it says: 'e:9-17-82' it means you joined (or backdated for all issues) on Sept. 17, 1982. It also means next issue is your last unless you join for another year. So clip the bottom third of the last page (with the address label on it) and mail it to Paul V. Skipski, our Secretary (his address is on the application form, last page), with your check for $15.00 U.S. (Canada and U.S.) or $25.00 (U.S.) overseas.

If your address label reads 'e:9-nn-83', it means you've already extended, with your membership dated from September, 1983. 'e:99-99-99' is a permanent membership, which does not expire. (The 'nn' above means any date in September.)

PLEASE send the address label; we must search the mail list to update, and you know what a slightly different name means in a search.... Leave off one initial and **#$%*--we do a special search. Be kind to Secretary and Editor.

****

TELE-COMMUNICATION : A BEGINNING

by John Frost, 7722 Fauntleroy Way, S.W., Seattle, Washington 98136

This is one in a series of articles to assist SPUG members in realizing the tele-communications potential of the SuperPet (SP). The article is written for the first-time user of the SP's RS-232 output port with a modem connection. In the first half of the article, we establish a basic configuration that permits a simple transfer of messages from the SP to/from Bulletin Boards, commercial data bases, other SP's or other communication-equipped home computers. As the prompts for telecommunicating with the SP are somewhat sparse, the article will 'walk you' thru a typical modem connection and describe the screen displays (or lack thereof). In the second half of the article we build upon these fundamentals and show how to reduce telephone connect charges by transmitting pre-stored data or messages directly from the microEDITOR (mED) to another computer (uploading).

In the last Gazette, several modem/cabling options were described. I can attest to successful telecommunications in the 6809 mode with a Lex-11 acoustically coupled RS-232 modem (currently advertised in the Heathkit catalog--and probably available from other sources), and either a standard interconnecting cable using all 25 pins, or a home-built cable described below.

Later articles will suggest use of a cable with fewer connections and a wiring modification at the end of the cable that connects to the SP. If you need to obtain a cable, I recommend you build the one described here, as it may be required when we explore telecommunications with Commodore BASIC 4.0. If you now have a cable with all 25 wires or the cable described in the last issue with 7 wires, by all means use the cable at hand and convert later if necessary.

The modified RS-232 cable between the SP and the modem should be wired so that the machine provides its own Carrier Detect, Clear to Send and Data Set Ready signals. These would normally come from the modem, however, for some reason they give some SP's indigestion (in BASIC 4.0) if the timing is not precisely controlled. To determine if your machine requires this cabling modification, simply run the 'supercom.bas' program on SPUG disk 1 (with or without a modem). If the machine crashes, odds are that your machine is sensitive to the problem; if so, I recommend cable jumpers. If the program executes to the point of displaying a menu, you may not need the special cable. If you do, here are the recommended RS-232 cable connections:

1. At the SP end of the cable jumper pin 4 to pin 5. (The SP's Request to Send signal provides the required Clear to Send). Similiarly jumper pin 20 to both pin 6 and pin 8. (Data Terminal Ready signal provides both Data Set Ready and Carrier Detect) KEEP ALL JUMPERS AS SHORT AS POSSIBLE.

2. With the jumpers in place, only three wires are required between the SP and the modem: Signal ground (pin 7), Transmitted Data (pin 2) and Received Data (pin 3). These cable modifications do not interfere with normal RS-232 operations with the 6809 processor.

Now, down to business! Connect the RS-232 cable to both SP's RS-232 port and to the modem (if you have the cable with the jumpers, make sure the jumpers are at the SP end). Apply power to the modem. If it has the following switches, set them as follows:

1) The ANSWER/ORIGINATE to ORIGINATE (if you are communicating with another SP or similiar computer, one of you will need to select ANSWER the other ORIGINATE. A bulletin board, host computer or commercial data base will expect you to be in ORIGINATE).

2) The FULL DUPLEX/HALF DUPLEX to FULL DUPLEX

If your modem does not have any of these switch options, don't be concerned; you will probably default to proper settings anyway. Power up in 6809 mode; select the 'setup' option from the screen menu. The screen will show the default communications options (overtype the ones you want to change). We will change only the baud rate (to 300) which is the standard for most applications with bulletin boards and data bases. Put the cursor over the 2 in 2400--enter 0300 and a <RETURN>. (The SP seems to require an entry in all four bit positions of the baud rate, hence the leading zero.)

On screen, the cursor will travel over all the displayed options (automatically entering your changes); the primary menu is again displayed. Select the 'edit' function and await the loading of the microEDITOR from disk.

Issue the 'talk' command. From this point on, all characters typed from the screen are sent to the modem (and the SP screen) and characters received from the modem are displayed. Now is the time to make your telephone connection, and upon hearing the carrier tone, activate the modem. (owners of acoustic modems will place the handset into the modem cups MAKING SURE THE MOUTHPIECE/EARPIECE ARE IN THE CORRECT ORIENTATION). Owners of direct coupled modems are referred to their instruction manuals. To evoke a response from a host computor, data base and the like, simply enter a carriage return or control character. A standard

ASCII CONTROL C can be sent from the keyboard using a SHIFT KEYPAD 3 (PF3), and a CONTROL B with PF2. From here on, follow the protocols dictated by the particular communication connection. If you have difficulty:

If no response is received from your communication partner, try operating the modem in its TEST mode. Keyboard entries should now be echoed back to the screen, giving you at least a warm feeling that your machine is correctly configured.

If your communication partner complains that his keyboard entries don't appear on his screen but show up correctly on yours, suggest that he switch his modem to HALF DUPLEX. (The SP doesn't normally provide the echo his machine requires).

If your SP double prints all keyboard entries when communicating with a host computor or data base service, it is an indication that your inputs are being echoed back from the host. Many hosts provide a command to defeat this echo. For instance, Commodore bulletin boards feature the DUP command and Compu-Serve offers a HALF command.

With the fundamentals behind us, let's explore a technique to cut down telephone connect time and charges. By storing a message or data in the mED prior to making the telephone connection, and then at the appropriate time directing the mED contents to the RS-232 port we have a simple form of up-loading. The data will be transmitted at the full 300 baud rate and not at a speed determined by your typing proficiency. NOTE: This is an especially powerful feature in light of recent Gazette articles on bringing a variety of files into the microEDITOR.

Here are the steps, assuming you have already addressed the 'setup' options:

1) Prepare a message or data in the mED by recalling a file from disk or inputting from the keyboard. (For your first attempt, I suggest only a few lines.)

2) Establish a modem connection per the earlier instructions with the 'talk' command. Don't worry about the mED contents scrolling off the screen or otherwise not being visible. The data will be available when needed.

3) Exit from the terminal mode with a STOP key entry. The screen will now show evidence of being in a '<HOLD>' mode with one cursor visible.

4) Enter a carriage return. Screen will now indicate the '<EDIT>' mode with two cursors visible.

5) Enter command 'put serial'. Not much will appear on the screen at this point altough a sharp ear might detect an acoustic modem processing the data. Also modem indicator lights will blink during this operation. This part of the operation is complete when the response 'serial - Lines transferred = xx' appears on the screen.

6) Quickly enter a 'talk' command; this returns you to the basic terminal mode and resumes the data exchange. Speed is somewhat important on this step as you do not want to lose any reply in response to your message.

In a following article we will explore the intricacies of a tele-communication link using the RS-232 port and Commodore 4.0 BASIC with the programs available

on the SPUG disk 1. These programs are designed to work with any of several Commodore bulletin boards across the USA and Canada and support a full range of up-loading and down-loading of basic 4.0 programs, Word-pro files, sequential files etc. The SPUG will offer a complete set of both terminal and bulletin board instructions.

* * *

A BUG IN READING TEXT FILES AND A CURE

In microFORTRAN, P.J. Rovero and others have run into a problem reading multiple data items on a single disk file line. One value reads okay; any attempt to get two or more will not read the last. If, for example, a file has the structure shown left, below, and you try to read it with 'format(3f10.1)', the last value on each line is not read. We know that SPET creates 'text' files on a disk as a default if any other type is not specified in the file command. 'p file', as an example, creates a 'text' file. All such files have variable size records and appear to end, P.J. says, at the last non-space printable character, which in the example above falls on byte 28; mFORTRAN won't read it because it 'knows' it cannot read 30 bytes in a 28-byte field (and a thirty-byte field is exactly what the format command wants: 3 f 10 calls for a thirty-byte read). P.J. has found two cures: The simplest is to add a printable character beyond the limit of the data to force a longer record size in the file as shown here: 5.6 6.7 8.8 x (the x falls at byte 31). Then, if you read with the same format command, you get the first thirty bytes okay.

```
 (line below is a byte index)
1234567890123456789012345678901
    5.6       6.7       8.8
    4.2       4.8       9.2
```

The second solution: specify a FIXED type file, where the record size is always set at a value independent of the the size of the data line. But you must use an extended filename: '(fixed:rec1).file-designator', a long and clumsy way, for both reading and writing. P.J. notes he has received several files with only one data item per line, apparently so written to avoid the problem described. We much prefer the use of the 'x'--it's far simpler. Thanks, P.J.

****

PROFILE : A SIMPLE PRINTER CONFIGURATION PROGRAM

A couple of months back, P.J. Rovero sent us a program to configure any SPET printer from the system disk (how do you set up your printer from DEVELOPMENT or the microEDITOR?). You may set it from the system disk before you load anything else with: profile <RETURN>. In 2 seconds or less your printer is set. You can revise the program easily (see the end) for any printer commands for any printer so long as the commands can be sent as a character string. Rovero sends two ESCAPE sequences: chr$(27),chr$(55) & chr$(27), chr$(106), with an ending '0' to tell SPET it's end-of-string. That sets up his BASE II printer (see cmdstring, below).

```
xref openf_
xref closef_
xref fprintf_
service_ equ $32

ldx #filemode
pshs x
ldd #filename
jsr openf_
std filecb
ldx #cmdstring
pshs x
ldd filecb
jsr fprintf_
ldd filecb
jsr closef_
```

The day it came in, we set up left margin on our printer with another sequence (see alternate cmdstring), which worked fine. You should be able to adapt the program to any printer with a few changed lines.

At first, we found trouble--we loaded profile at $7000 and overloaded languages on top of it, thinking that the languages wouldn't mind. They did mind; we ran into all sorts of problems. Then we stuck it in at $0a00, at the very bottom of user memory--and more problems. Finally,

```
ldb #0
stb service_
puls x,y
ldx #0
ldy #0
rts
```

we stuffed it in $7f00 and overwrote it each time with PIRQ, which sets mem-end at $7f00. That works well, unless you don't use PIRQ. P.J. came up with an alternate method: load in Bank 1 of the upper 64, which is surely overwritten by any language or facility you subseqently load from the language disk. No troubles to date. So you have two methods which work. Loading in Bank 0 should be equally good.

```
filecb    fdb 0
filename  fcc "ieee4"   ; change to 'printer' or 'serial' if needed
          fcb 0
filemode  fcc "write"
          fcb 0
cmdstring fcb 27,55     ; sends ESCAPE 7
          fcb 27,106,0  ; sends ESCAPE j and string-end 0
          end
```

To show how easy it is to adapt the program to another printer, here's the cmd-string for DIABLO 630 (COMMODORE 8300P):

```
cmdstring fcb 27,13,80  ; sends ESCAPE CR P to zero DIABLO left margin
          ; line below sends ten spaces and ESCAPE 9, CR, and terminating 0's
          ; to set left margin at 10 (three 0's for odd number in string)
          fcb 32,32,32,32,32,32,32,32,32,32,27,57,13,0,0,0
          end  ; for reason unknown, DIABLO won't set on: "          "
                          *          *          *
```

```
"profile"
bank 1                          ; profile.cmd, the linker file
include "disk/1.watlib.exp"
org $9000
"profile.b09"
```

Thanks to P.J. for another useful SuperPET tool.

****

We continue the article on text-processing in SuperPET from the last issue:

CONVERTING BASIC 4.0 PROGRAMS FOR WORDPRO

If you'd rather bring your BASIC programs into WordPro, you can do it with another program on the SPUG disk: CONVERTPRO. You can also use LIST-9 to convert the cursor commands to English before you bring the program to WordPro. Let's take the procedures one at a time. (1) To bring BASIC programs into WordPro: 'dload' the program, then put it to disk as a program file with the first line, left. As soon as the drive stops & cursor returns, enter the second line. You may then enter: new <RETURN> and 'dload' CONVERTPRO. Run the program; follow the prompts. The program always 'gets' from one drive and 'saves' to the other, to whatever filename you may enter; it automatically scratches the temporary file you created with the command above, left. The conversion prints to the screen as well as to disk (the screen printout is not in final format, but rather a progress report). As with LIST-9, you may choose the character count of lines (from a minimum of 30 to a maximum of 80). The file created by CONVERTPRO is then a WordPro file, and may be integrated with WordPro text. We rewrote CONVERTPRO from a program written by Neil Harris in Jan/Feb '83 COMMODORE, making it more automatic and easier to run. We thank Neil for the basic approach.

```
open 1,8,2,"0:filename,p,w":cmd 1:list
print#1 : close 1
```

(2) To convert cursor commands into English and bring BASIC programs into WordPro. Get LIST-9 with: load '0:list-9',8 (if it's on drive 0). Then enter: new

<RETURN>. Next, 'dload' the program you wish to convert, and issue the commands shown to the left. The 'n', again, is the number of characters wanted per line. The file having been put on disk, enter: new <RETURN>, load CONVERTPRO, and run it. It will convert the disk file to a WordPro file on the other drive. There is one glitch (and we hope someone out there can fix it): left and right brackets '[ ]' appear as a cross and vertical bar in WordPro. It's easy to find them and to replace them with brackets. We tried a dozen solutions (and so did Gary), but nothing worked. We decided to publish anyway, since 99% conversion is better than none. If you can fix the bug, let us know how.

```
open 5,8,4 "1:filename,p,w":poke 78,n:cmd5:sys 40539
print#5 : close 5 : rem Assumes saving to drive 1
```

Gary notes that the original code for LIST-9 was written by the Code Works for early PET ROMs; he has permission to modify from the version which appeared in CURSOR #5 many long moons ago, and to publish it. Thanks, Code Works.

* * *

A LITTLE TRANSLATION OR REVERSE ENGLISH

If you're not interested in integrating text and programs in mED, are you interested in translating BASIC utility programs to mBASIC? Or in getting mPASCAL or mFORTRAN or mCOBOL programs found in a WordPro file up and running without retyping them? Or in converting programs you picked up by a 6502 telecomm to 6809 programs? In short, are you interested in the reverse English of the process we've been describing? You can convert any WordPro or Wordcraft file to an ASCII file, pull it into the mED, strip off the text, and file the programs it contains as SuperPET sequential files. Then you can load whatever language you want and RUN them. No retyping. We've been doing it for a couple of months. ANYTHING in 6502 (except ML programs) can be brought into 6809 and either translated or run. See John Frost on telecomm, this issue, for the significance.

* * *

CONVERTING WORDPRO AND WORDCRAFT FILES FOR USE IN 6809 MODE

The basic process is simple: convert files to ASCII files. Both WordPro and Wordcraft provide means to do so. Following are the instructions for WordPro. When you load WordPro, and get the starting menu, pick whatever options you normally use. Then proceed as follows:

Status Line: (Top of Screen, Left)	Your Action:	Comments:
	<CONTROL> o d <RETURN>	For a page on screen.
	<CONTROL> o g d <RETURN>	For a global disk file.
CBM, ASC, OR PRINTER?	Press 'a'	No <RETURN>
ASC FILE. DRIVE #?	Press '1' or '0'	No <RETURN>. Location of new ASCII File. (1 picked)
ASC FILE:1:	Enter New Filename	Hit <RETURN>

(A screen page is now converted. Global files will give one more prompt:)

GLOBAL?	Enter Filename, 1st Page of Global File to Convert.	Hit <RETURN>

That's it. When one of these files is called into mED, it will have a quotation mark at the beginning of each line. Get rid of it with the search and replace command, at command cursor: *c/%↑"/ <RETURN>. The command means: find, at start of line, a quotation mark, and replace it with--nothing. It's a delete command. The command is slow, so be patient. Suddenly all the quotes will be gone, and all text pulled one space left. On very long text, you may wait several minutes.

You will find all underlined material in reverse video (reverse field). Underlined words or phrases cannot be underlined in mED, but plain underlines, as in forms or as in library function fptchar_, are easily entered with the left arrow key. The reverse field tells you where to enter them. The reverse field characters print as regular characters to DIABLO 630, but you may have to overtype with plain characters for other printers. The best solution: don't underline text you plan to convert if your printer can't handle reverse field characters.

* * *

AN EXTRA GOODIE: A DIRECTORY SORT

The directories on a 4040 are bad enough, but a directory of a full disk in 8050 is a nightmare. We shudder to even think of finding the file we want on a 5-meg hard disk. Well, there's a way. On the SPUG disk offered last issue ($10.00, no disk, to Secretary Paul Skipski) is a machine-language alphanumeric directory sort, DIR SORT.S. It will alphabetize any directory in SPET, and runs in BASIC 4.0. It was designed for WordPro files (haven't tried it on Wordcraft). Run the program in BASIC 4.0 (just follow the prompts), and it prints out an alphabetized directory to printer. The program also files an index to disk (creatively named 'index'), which you can read on screen or put to printer in WordPro (but not in BASIC 4.0, or in 6809 directly). We don't have a program or way to convert the file directly into a sequential file for mED without WordPro. (Somebody please write one!) With WordPro, of course, you simply convert the disk file 'index' to an ASCII file the mED can read in 6809, as we note above.

We hope somebody will convert DIR SORT.S to a 6809 program, with an appropriate 6809 assembler shell sort (the one in 6502, written by Gary Ratliff, inserted in the sort by Gary, and found on the SPUG disk as SHELL, is fast indeed). Note: the program is set up to work with a DIABLO 630 (COMMODORE 8300P) printer. Revise for yours. No, you need enter nothing in the monitor. The sort loads SHELL out of program. Gary's fast SHELL sort has cut run time in half. Loading directories into RAM for sorting is now the slow process.

****

HANDLING INTERMIXED APL AND ASCII FILES

So long as you don't have overstruck APL characters, you can bring any sequential APL file into mED and put it into ASCII text. The problem, of course, occurs at printout to printer. If you can change printwheels or adjust your printer for APL, simply page manually in mED down to the APL material, change your printer, and then print the APL stuff. Then switch the printer back to ASCII and proceed.

Overstruck characters can't be printed from the mED; they must be printed either from disk or from APL. The program PRINTALL, on SPUG disk 1, runs in mBASIC, and prints any sequential 6809 file. If that sequential file (like this one), has a line in it which reads AAAPL at the left margin, the program stops and tells you to set your printer up for APL, opens the APL file, prints it from disk, and then stops when it reaches end-of-file, and tells you to reset your printer for ASCII. There are no limits on the number of APL files which may be so intermixed. The APL lines sent to printer are counted and paged just as though they were part of the 6809 sequential file. We used the

AAAPL

method to do Steve Zeller's column, footers and page numbers included, for the last few issues, and with changes still use it. Full data is on ISPUG disk 1.

* * *

FEEDBACK ON PRINTALL — Those with SPUG disk 1 should add to PRINTALL line 1017:
1017 if header$='' and (A or WA) then call change(3)
We neglected to call for a printwheel or printer change at end ASCII footer and with no following header, whilst in APL. Sorry. Second, we have reports that D$ (chr$(10)) in proc footer causes graphics to print instead of ASCII on the left$ of a footer on some Commodore printers. Solution: change D$ (chr$(10)) to CR$ or chr$(13) in proc footer. Problem disappears. Gary Ratliff found the problem and the answer.

* * *

GIANT VIRTUAL AND LITERAL FILES — For lack of space, we'll cover this material next issue. Suffice it to say that we now have the capability to review, page, and output to disk file and to printer all of the Encylopaedia Brittanica in one, single, long pass (if we cared to), and retain the ability to bring any part of such a giant file into the mED later if we want to revise, review, or print part of it. The only problem in so doing: being thickheaded in previous attempts.

****

DISK I/O ERROR CHECKING
Comments by John Spencer — Dr. Spencer has sent us some useful notes on I/O error checks, and comments on 'proc ds', published in the November '82 Gazette. Proc ds provides error information only when the red light on the disk drive is on, and only when the system itself has not cleared the error. Most of the time, you get an '00,OK,00,

00' message--but not always. File something to disk from screen, and remove the disk while it's being filed to get a red light. Then call 'proc ds'. You'll get '21,READ ERROR,38,03,0', which is accurate enough (among other things, 'no disk present'). Proc ds always turns off the red light. (Very often, a directory call also will turn off the red LED, if the error has been corrected by the system.)

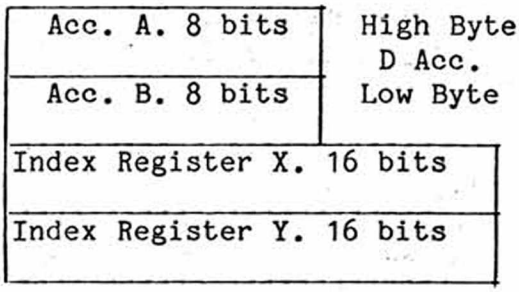
Dr. Spencer notes that you can ask for io_status$(1:2) when you want to know only the error number; and that io_status$, being an intrinsic function, does not behave as an ordinary string variable. If you set aa$=io_status$, aa$ will turn out NOT to be the original io_status message; it is garbled when translated. If, for example, there is no disk in a drive, io_status$ = '74, DRIVE NOT READY, 00 00', but aa$ = 'file not open'. Dr. Spencer suggests a solution which may be of use in all the Waterloo languages: The contents of io_status$ are stored in the form of characters in memory locations 768-792; io_status in 1 byte at location 106. If you wish to determine the disk error number, convert the first two bytes (above, left). To obtain the corresponding DOS error message, peek and convert locations 771-785.

```
ioerr%=value(chr$(peek(768)+chr$(peek(769)))
```

****

BITS BYTES & BUGS

by Gary L. Ratliff, P.O. Box 829, Sanatorium, Mississippi 39112



In the last installment, the address indexing insructions may have been hard to follow: lda, x+ and sta ,y+. Both are special forms of indexing which include auto-increment of the two index registers, x and y, which point to locations you'll use. Note the diagram at left. Initially, the x register points to the first character of our message and the y register, in turn, points to the first screen address. Let us assume for discussion that the message begins at location $1024. The instruction lda ,x+ loads the data from location $1024 and increments the x index register so it points to $1025. When the sta ,y+ instruction is executed the contents of the a register are stored at memory location $8000 and the y index register is incremented to point to $8001.

Clearly the indexed addressing mode with auto increment is a very powerful method for processing text. However, the 6809 has many more indexed addressing modes which we will explore in this installment.

There is no reason why we must increment the index registers. It is often much faster to write a loop which decrements the index registers. (This is useful more often with the 6502 which has index registers which are only 8 bytes in length). If we decrement the y register, we can send the message backwards. To do so, first link and execute the program. Note that the final value of the y register is $80f8. Load the file text.asm into the EDITOR; change the value assigned to screen from $8000 to $80f8. (Find the line, at command cursor, with: /equ <RETURN>). Then find sta ,y+ and change it to sta ,-y. The program is complete; save it as textb.asm. Then recall your .cmd file and change all 'text' to 'textb', save it, assemble and link, as explained in the last installment. Load the program in the monitor with >l testb.mod, and run it with >g 1000, and watch the message appear backwards on the screen.

In the 6809, you may combine the A and B accumulators into a single D accumulator (a distinct advantage over the 6502). This means that we could just as easily send the message two bytes at a time. We need to be certain, however, that we can end our program loop with the needed zero. If we say ldd ,x++, then the next two bytes of the message will be loaded into the combined D accumulator. The ++ increments the x register by two instead of one. We also change the sta ,y+ instruction to std ,y++. Finally, we satisfy the requirement that the D accumulator contain zero by adding three fcb 0 instructions. If the message contains an even number of letters when the end is reached, the next cycle of the loop causes the D accumulator to contain zero. If, however, an odd number of letters are in the message, the first zero will be in the B accumulator and on the next pass the second and third of the fcb 0 instructions will stop the execution of the loop. I suggest you load text.asm, make the changes, assemble, link, and run. Note that you do not have to erase the old files. SPET will overwrite them.

If you wonder why the odd or even number of characters does what I note above, remember that each byte holds one character. If the number of characters is odd, the low byte (Accumulator B) will, at end of message, hold a zero, but the high byte (Accumulator A) will hold the code for the last character.

Note we could use indexed addressing to send not the completed message but only every other letter. If we change the std ,y++ instruction to sta ,y++ we will send only the odd numbered characters of the message and if we use stb ,y++ we will transmit only the even numbered characters. Alter your files and try it.

Although sending text messages backwards and sending only every other character of a message may not be immediately practical, you will gain a feel of how indexed addressing with auto-increment works.

There is real power in the DEVELOPMENT system, which affords users easy ways to perform routine tasks by using the built-in functions of the Waterloo library. To use a library routine, you need only determine the required parameters and pass them to the routine. All parameters greater than 1 are placed on the stack. The highest numbered parameters are placed first followed by lower ordered parameters. The programmer is responsible for removing these parameters from the stack after the routine is finished.

Looking through the list of library routines we find that the routine copy_ will move the contents of memory from one location to another. Clearly, this is what we have been doing with our string. We close this installment by modifying the program text.asm to use the built-in routine copy_ [All library routines end with the underline character. On the 6809 side this is achieved with the back-arrow key.]

Copy_ requires that three parameters be passed to the routine. P3 is the length to copy, P2 is the address of the destination (we still want to send the message to the screen), and P1 is the starting address of the memory we want to copy. In all library routines P1 is passed in the D accumulator; the other parameters, P3 and P2, are placed on the stack. The assembly program to accomplish this is presented below:

```
; send message to screen using library routine: copy_
screen equ $8000
       xref copy_  ; all library routines must be placed first.
```

```
        ldd # len    ; this is P3, the length parameter.
        pshs d       ; this pushes the contents of D on the stack.
        ldd #screen  ; this is P2, the destination parameter.
        pshs d       ; so put it on the stack.
        ldd # msg    ; this is P1, the starting address parameter.
        jsr copy_    ; this performs the desired copy.
        swi
msg     fcc 'This is the text we wish to send to the screen.'
len     equ * - msg ; * is the current location pointer.
        end
```

Save this as program: move.asm, and save the following as move.cmd.

```
'move'
org $1000
include 'disk/1.watlib.exp' ; this looks up address of library routines.
'move.b09'                  ; put your language disk in drive 1.
```

Because two months is a long time to wait, starting with this issue, we announce the error in the code on the last page of the Gazette, figuring you are earnest enough to learn by trying to figure it out for yourself.

In the next installment, we'll explore how to optimize a program using the library routine length_. [Ed. the mystery of that cryptic filename '.b09' is now explained; the Development package for the 6502 codes it '.b02'. Last two digits of the microprocessor number comprise the codename. But why the 'b'?]

****

PRINT DIRECTORY [We conclude Dr. Spencer's article from the last issue.]

In line 240, we use 'on conv' to handle non-numeric input when we want a number for a disk drive. We could input a number directly in line 285, but if any character other than a number, +,-, or the decimal point are typed, microBASIC then flashes a 'Redo from Start' message which bewilders the untrained. (And we cannot accept +, -, or the decimal point!). Instead, we ask for string input of the drive number; if we fail to convert dr$ to dr% in line 290, the program resumes at line 295 on 'resume next', and we loop back to line 285 to try again to get a string number of 0 or 1.

If the user forgets to put a disk in the specified drive (or leaves on a write-protect tab) we have another problem, handled by the 'on ioerr' at line 240. We can detect the presence of a disk in a drive in several ways; here, we do it by attempting to open file 'zz' for output. So long as we have no file 'zz' on that disk, no harm is done. If a disk is absent or a write-protect tab is present, the 'ioerr' routine flashes a screen message and waits for the user to correct the disk error and to signal when he is done. Since nothing has been written to file 'zz' (except on the directory tracks), we may get access to full disks with this method.

Although disabling the keyboard in a program such as this is overkill without a doubt, we often encounter practical situations where it is necessary; e.g., in chaining programs together. It seems wise to set up an 'on attn' line which will disable STOP before the keyboard itself is enabled, so that no STOPs can creep into the keyboard buffer by inadvertence when chaining. Some programs require no

keyboard input and may run unattended; we can protect them from the curious by a keyboard disable; similarly, we can avoid ambiguity on 'resume/resume next' if we write long, open-ended strings to disk files. Last, we may increase program security. We access a program by chaining from a driver program which disables the keyboard. In the program itself, we may chain back to another program to reset the keyboard or we may use simulated immediate mode to implement a 'clear' statement from the screen before the program finishes. Those who know only the 'list' command cannot access such a program.

We may wish to stop or pause a program by an action more deliberate than touching STOP. One solution is shown at left, below; to employ it, first touch STOP, which puts you into 'on attn'; there you cycle in the 'get' at line 213 until a key is touched. If you now press SHIFT/RUN the program stops; a touch of any other key resumes program. You may substitute the ordinal of any other key for SHIFT/RUN. The lines above may be placed in PRINT DIRECTORY if you want to be able to STOP the program. [Note from the Crash Wagon, written on a bandaid: Use it! Ed.]

```
210 on attn
213   get a$ : if a$='' then 243
215   if ord(a$)<>2 then resume ! or resume next
217   stop  ! or pause + resume
220 endon
```

We simulate immediate mode in line 360 of this program by stuffing a series of characters into the keyboard buffer just prior to the 'stop' at 365. The characters clear the screen, print the directory command, and enter one or more carriage returns (CR's). The first CR executes the 'di' command; the subsequent CR's scroll to the next new directory page, the number of CR's increasing by one each cycle. The last entry in the keyboard buffer is the immediate mode command 'goto 375', followed by another CR, which executes the 'goto'. The programs then reads the directory lines into AA$; on the first page of the directory, AA$ begins to fill just after the 'di' command is read. On the last and usually partial page, AA$ reads no entries until it encounters a new line. (We use the function 'idx' to make sure we the lines are new.) The process stops when the BLOCKS FREE is found at end of directory.

The method may be used to determine the number of disk blocks free before we attempt to write to disk. PRINT DIRECTORY also adds the date in brackets if the date is set, either from the microEDITOR or as noted in the Gazette in Volume 1, at page 49.

[Note: Dr. Spencer furnishes replacement lines, shown below, to speed up PRINT DIRECTORY. As the giant string of directory entries grows, idx search of it gets slower and slower. The revision, below, eliminates the delay.]

```
410   print #131, CR$ : aa%=cursor(aa%+80) : linput '',aa$ : if gg% then 425
425   gg%=1 : if aa$ = 'goto 375' then 445
445   ncr%=ncr%+1 : gg%=0 : goto 355
```

****

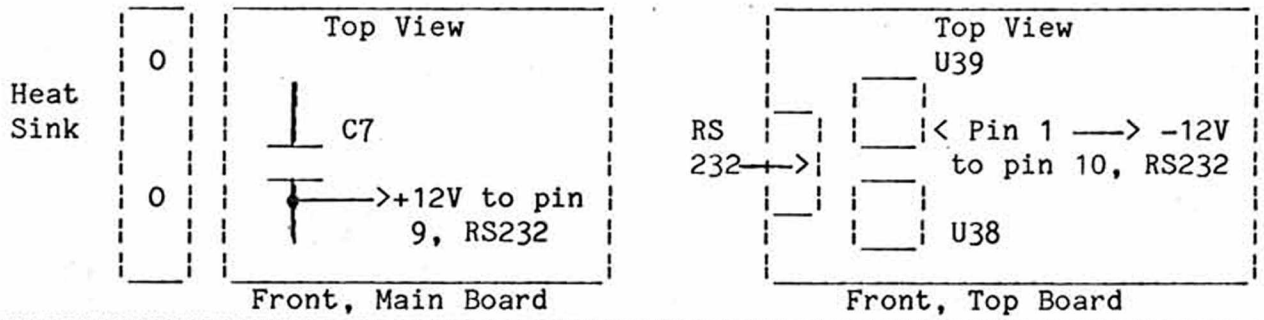
COPING WITH LONG RUNS OFF THE SERIAL PORT

by Don Gilbreath
Technical Support Manager
Commodore Dallas

Stock SuperPETS are quite adequate for short serial destinations of one-quarter mile or less. In large buildings, however, where runs of serial lines need approach one to two miles, 'line drivers' are the only solution. You will find quite a few good line drivers on the market, not to mention that for the hobbyist they're quite easy to construct. The only obstacles: pins 9 and 10 of

the RS-232 connector, which are intended for +12 and -12 volts, respectively. Unfortunately, Commodore failed to bring these lines out, and to use commercial line drivers you must address the problem.

My solution: find the necessary voltages on SPET and bring those voltages out to pins 9 and 10 of the RS232 connector. +12 volts can easily be reached on the closest capacitor to the front regulator on the main motherboard (lower board). See drawing below. Jumper one lead of capacitor C7, as shown, to pin 9 on the RS232 connector in SPET. You get -12 volts at pin 1 of U39, which is the existent 1488 driver sitting next to the RS-232 connector. Jumper this pin to pin 10 on the RS-232. You can now get 9600 bps at a distance of two miles. [Short, clear, and useful. Thanks, Don. Ed.]



****

USING HOSTCM : A REVIEW
by P. J. Rovero
SMC Box 1610, NPGS
Monterey, CA 93940

HOSTCM is a Waterloo mainframe communications program which directly interfaces SuperPET and other micros which run Waterloo software, and is available for the IBM VM/CMS and DEC RSTS/E operating systems. The version reviewed runs on an IBM 3033, and all implementations should have the same command syntax, though host filenames will vary.

Why link SPET to a mainframe? Programs written on SPET are easily ported to the mainframe and compiled, particularly on mainframes which run Waterloo compilers (WATFIV for FORTRAN and WATBOL for COBOL). Those who must submit work on a mainframe or must access large amounts of memory, plotters, and high-speed printers can use the SPET-mainframe link, which is simple.

When HOSTCM runs on the mainframe, that mainframe looks like 'just another disk drive' to SPET. Almost any file-oriented mED command can be sent to the mainframe with a prefix of 'host.' (with minor restrictions on full specification of filenames and filetypes).

SPET recognizes three filetypes: TEXT, VARIABLE, and FIXED. TEXT files are variable-sized records of printable characters--normal SPET source files, to which SPET defaults as a filetype. VARIABLE files have variable-sized records which may contain non-printable characters. The 6809 .mod files (load modules) created by the DEVELOPMENT package are VARIABLE files. FIXED files have a fixed record length and may contain non-printable characters. Data files in mFORTRAN and in mBASIC are often of FIXED type.

The mainframe filetypes have different names but correspond to one of the SPET types. When the mainframe filetypes are other than TEXT, you must know how to specify the proper filetype. Source code files on the IBM UM/CMS operating system correspond to SPET's TEXT type; data files for FORTRAN must correspond to card format and should be specified as '(f:80)host.' Binary files (6809 or IBM 360 object code) must be specified by 'v' for VARIABLE.

Hooking up to HOSTCM is simple. The SPET operator must first execute SETUP, and set baud rate and other parameters to the host's requirements (the defaults are for UM/CMS, so I change only the baud rate). Then you invoke the mED and execute the 'talk' command to turn SPET into a dumb terminal, then dial and log onto the host, and then invoke HOSTCM, which notifies you that the program is executing. Press the STOP key to return to the mED, where you are now ready to roll. Here are a few examples:

Put the current file to the host with filename 'test fortran'	p 'host.test fortran'
Get the host file 'test2 pascal' into the mED	g 'host.test2 pascal'
Copy the SPET file 'stock_d' to the host as a fixed data file	copy stock_d to '(f:80)host.stock d'
Copy a 6809 object module to the mainframe	copy prof68,prg to (v)host.prof68
Copy 6809 object module from host to SPET	copy (v)host.prof68 to prof68,prg
Get host file directory	dir host

Note that filenames which contain spaces must be in quotes, and that 'prg' must be specified for program files on disks. SPET defaults to 'seq' files unless it is otherwise directed. Had 'prg' been missing from the last command, SPET would have filed the object code as a sequential file (which will not load). Of course a copy command, from disk to disk, in SPET alone, does not suffer this problem.

You turn HOSTCM off when you issue the 'talk' command and send a 'q' with a <RETURN>, log off the host, and return to the mED with the STOP key. Pretty simple and straightforward. HOSTCM works as advertised; Waterloo Computing Systems produces quality products and supports them. HOSTCM is simple to use and transfers files reliably to and from the host, but transfer is slower than the baud rate because errors are checked and bad data retransmitted, and because binary files are sent in a slow, exploded format. Files transfer in a couple of minutes but the transfer is free of error. My only problem to date is my inability to directly load a 6809 object module (a '.mod' file) stored on the host into the micro from the main menu (maybe I just haven't figured it out yet); a minor problem, for you can later load it from disk. Overall, HOSTCM is excellent. [P.J. disclaims any relationship with Waterloo Computing Systems, except the status of a happy customer. Count me in, too. Ed.]

(C) 1983 ◇◇◇ THE APL EXCHANGE ◇◇◇ STEVE ZELLER

At the behest of our Editor, I diverge from the planned discussion of branching within functions and instead consider some routines that will move APL functions to and from disk. While a workspace represents an important part of the APL implementation, use of the)COPY or)PCOPY commands to draw functions from disk into the current WS is not efficient. The WS on disk must be read serially in order to find the needed function and, as George Cordhai has pointed out, there is some disk storage "overhead" involved as well. An alternative is to employ

APL sequential files to store functions individually, and then read them in as needed. With this approach, we let DOS do the looking for us.

The two functions presented here will serve this role. They are very similar to examples contributed by other APL'ers in SPUG. Both use a three part file naming scheme. The first part, or prefix, is meant to represent membership of the function in some larger collection of functions and data. For example, with the EDA disk announced last time, I prefixed all files names with: 'EDA.' In addition, all names of files containing APL functions end with: '.AFN'. Depending on the length of the prefix, this leaves about eight characters in the filename for the function's name. (The actual name of the APL function can, of course, be longer and need not correspond exactly to this portion of the file name.) These functions are shown below:

```
      ΔDISK       ⍝CREATE A 'SYSTEM VARIABLE' THAT INDICATES WHICH DISK
DISK/0.           ⍝YOU ARE 'PUTTING' AND 'GETTING' FUNCTIONS TO AND FROM.
      ∇ΔGETF[⎕]∇
[  0]    R ← PREFIX ΔGETF FN ;FFN;NF;⎕IOERRSTOP  ⍝USE THIS FUNCTION TO GET
[  1]   ⍝GETS FUNCTION STORED AS APL SEQ. FILE   FUNCTIONS ALREADY ON DISK.
[  2]   ⎕IOERRSTOP←0                  ⍝THE FUNCTION ARGUMENT 'PREFIX' MIGHT
[  3]   FFN←ΔDISK,PREFIX,FN,'.AFN'     ⍝BE A GLOBAL IDENTIFIER UNDER WHICH YOU
[  4]   FFN ⎕TIE NF←ΔNEXTNF            ⍝HAVE GROUPED A NUMBER OF FUNCTIONS.
[  5]   →(0≠ρ⎕STATUS)/ERR              ⍝ALTERNATIVELY, IT MIGHT BE EMPTY (⍳0).
[  6]     ⎕FX ⎕READ NF                 ⍝ALL FUNCTION FILE NAMES END WITH '.AFN'
[  7]     R←1                          ⍝THE FUNCTION, AS TO BE NAMED IN THE WS,
[  8]     →EXIT                        ⍝IS GIVEN BY THE FUNCTION ARGUMENT, FN.
[  9]   ERR:⎕STATUS                    ⍝EXAMPLE: 'SZ.' ΔGETF 'ΔNEXTNF' WOULD
[ 10]     R←0                          ⍝TIE A FILE NAMED: 'SZ.ΔNEXTNF.AFN' ON
[ 11]   EXIT:⎕UNTIE NF                 ⍝DRIVE ZERO OF UNIT EIGHT AND ESTABLISH
      ∇ΔNEXTNF[⎕]∇                     ⍝IN THE WS AS: ΔNEXTNF. NOTE: SUCCESSFUL
[  0]    NF ← ΔNEXTNF                  ⍝GET WILL RETURN THE BOOLEAN VALUE OF 1
[  1]   ⍝ RETURNS NEXT AVAIL. FILE NO. ABOVE 10.  THIS FUNCTION IS USED TO
[  2]   NF←1+10⌈⌈/0,⎕NUMS  ⍝CREATE A FILE NUMBER W/O HAVING TO KNOW WHAT IS
                           ⍝ALREADY BEING USED IN THE WS (AND IS IN ⎕NUMS).

      ∇ΔPUTF[⎕]∇       ⍝THIS IS THE COMPANION FUNCTION TO 'ΔGETF'.
[  0]    R ← PREFIX ΔPUTF FN ;NF;FFN;⎕IOERRSTOP
[  1]   ⍝ STORES FUNCTION ON DISK, AS APL SEQ. FILE
[  2]   →(3≠⎕NC FN)/ERR0               ⍝BOTH FUNCTION ARGUMENTS, PREFIX AND FN,
[  3]   ⎕IOERRSTOP←0                   ⍝WORK AS IN ΔPUTF.
[  4]   FFN←ΔDISK,PREFIX,FN,'.AFN'     ⍝'ΔPUTF' WILL NOT LET YOU OVERWRITE AN
[  5]   →(1=ΔISONDISK FFN)/ERR1        ⍝EXISTING FUNCTION ON DISK. IF YOU WANT
[  6]     FFN ⎕CREATE NF←ΔNEXTNF       ⍝TO REPLACE THAT FUNCTION, USE ⎕ERASE
[  7]     (⎕CR FN) ⎕WRITE NF           ⍝TO ERASE THE FILE AND THEN USE ΔPUTF
[  8]     →(0≠ρ⎕STATUS)/ERR2           ⍝TO PUT THE NEW VERSION ON DISK.
[  9]       R←1                              ⍝EXAMPLES:
[ 10]       →EXIT                            ⍝  'SZ.' ΔPUTF 'ΔNEXTNF'
[ 11]   ERR0:FN,' IS NOT A FUNCTION IN WS'   ⍝  ⍳0 ΔPUTF 'GARBAGE'
[ 12]     →(R←0)                             ⍝  ⍳0 ΔPUTF 'GARBAGE'
[ 13]   ERR1:'FILE ALREADY ON DISK..'        ⍝A SUCCESSFUL 'PUT' RETURNS A
[ 14]     →(R←0)                             ⍝BOOLEAN OF ONE. NOTE THAT I DO
[ 15]   ERR2:⎕STATUS                         ⍝NOT ALLOW NON FUNCTIONS TO BE
[ 16]     R←0                                ⍝SAVED TO DISK WITH THIS FUNCTION
[ 17]   EXIT:⎕UNTIE NF  ⍝⎕IOERRSTOP IS LOCALIZED IN THE HEADER, SO THAT IT IS
                        ⍝UNITY AFTER RETURNING FROM THIS FUNCTION.
```

```
      ∇ΔISONDISK[□]∇
[  0]    R ← ΔISONDISK FN ;NF;□IOERRSTOP
[  1]   ⍝CHECKS TO SEE IF FILE IS ON DISK
[  2]   □IOERRSTOP←0          ⍝RETURNS A LOGICAL 'YES' IF THE FILE ALREADY
[  3]   FN □TIE NF←ΔNEXTNF    ⍝EXISTS. NOTE THAT THERE IS NO PRESUMED FILE
[  4]   →(0≠ρ□STATUS)/ERR    ⍝NAMING CONVENTION WITH THIS UTILITY. THIS
[  5]     R←1     ⍝TRUE       ⍝MEANS THAT THIS FUNCTION CAN BE USED IN A
[  6]     →EXIT               ⍝NUMBER OF CONTEXTS.
[  7]   ERR:R←0   ⍝ FALSE
[  8]   EXIT:□UNTIE NF
```

Note that this naming scheme becomes very useful with MicroPIP (see next issue for review).

Another source of confusion centers around the use of overstruck characters in APL. In the example below, we see how we can move back and forth between internal and external representations of APL characters. The example is followed by functions that can be used to send matrices of characters to the printer. The final example presents a modification of Jim Swift's "dump" routine that allows one to capture a function listed on the screen in a character matrix. This matrix can then either be printed or stored on disk.

```
A SIMPLE EXAMPLE OF THE USE OF □XR AND □IR:
    ρC1←,'⍵'     ⍝THREE KEYSTROKES ARE CONVERTED BY THE TERMINAL MONITOR
1                ⍝INTO A SINGLE CHARACTER. USING "SCAN", WE SEE THAT IT IS
    □AV⍳C1       ⍝THE 20TH CHARACTER IN THE APL CHARACTERSET. CHECK THIS BY
20               ⍝APPENDIX C, P. 105, IN THE WCS APL MANUAL. THIS CHARACTER
    ρC2←□XR C1   ⍝CAN BE PRODUCED BY YOUR PRINTER BY GOING BACK TO THE THREE
3                ⍝CHARACTER SEQUENCE USED IN CONSTRUCTING THE "AXLE". WE CAN
    □AV⍳C2       ⍝DO THIS WITH THE SYSTEM FUNCTION, "□XR", WHICH PRODUCES
80 9 64          ⍝THREE ELEMENT VECTOR CONSISTING OF (1) A "CIRCLE", (2) A
    ρC3←□IR C2   ⍝BACKSPACE, AND (3) "BACKSLASH". SIMILARLY, WE CAN TRANS-
1                ⍝IN THE OPPOSITE DIRECTION WITH THE SYSTEM FUNCTION, "□IR".
    □AV⍳C3       ⍝INDEED, THE TWO FUNCTIONS, OPERATING TOGETHER, HAVE NO
20               ⍝EFFECT.

EXAMPLES OF ROUTINES TO PRINT CHARACTERS.
      ∇PRINT[□]∇
[  0]    PRINT STUFF ;NF
[  1]   ⍝SEND OUTPUT TO THE PRINTER
[  2]   'IEEE4' □CREATE NF←ΔNEXTNF
[  3]   STUFF □PUT NF              ⍝THIS WORKS FINE FOR DATA, WHERE WE DO NOT
[  4]   □UNTIE NF                  ⍝ASCII-APL OVERLAY CHARACTERS.
      ∇PRINTXR[□]∇
[  0]    PRINTXR STUFF ;NF
[  1]   ⍝SEND OUTPUT TO PRINTER, USING TYPEWRITER-PAIRING
[  2]   'IEEE4' □CREATE NF←ΔNEXTNF
[  3]   (□XR ,STUFF,□TC[□IO+6]) □PUT NF   ⍝WE NEED SOMETHING LIKE THIS FOR
[  4]   □UNTIE NF                         ⍝CHARACTERS. NOTE THAT "□XR" WILL
      ∇RESHAPE[□]∇                        ⍝NOT ACCEPT A MATRIX ARGUMENT. TO
[  0]    R ← N RESHAPE M ;NR;NW           ⍝GET AROUND THIS PROBLEM, WE FIRST
[  1]   ⍝RESHAPES CHARACTERS TO WIDTH N   ⍝APPEND LINEFEEDS TO THE MATRIX A
[  2]   →(1<ρρM)/MATRIX                   ⍝THEN "RAVEL" THE RESULT.
[  3]     NW←⌈(ρM)÷N
```

```
[ 4]     R←(NW,N)ρ(NW×N)↑M                    ⍝THE FUNCTION ¨RESHAPE¨ CAN BE USED
[ 5]      →0                                  ⍝TO RESHAPE CHARACTER VECTORS OR
[ 6]    MATRIX:NW←⌈(¯1↑ρM)÷N                  ⍝MATRICES THAT ARE TOO LONG FOR THE
[ 7]     NR←1↑ρM                              ⍝PRINTER. EXAMPLE:
[ 8]     R←((NW×NR),N)ρ(NR,(NW×N))↑M          ⍝    PRINTXR 80 RESHAPE XX
```

FINALLY, I JUST COULDN'T RESIST MODIFYING JIM SWIFT'S GREAT SCREEN DUMP ROUTINE SLIGHTLY IN ORDER TO ALLOW ME TO ¨CAPTURE¨ THE SCREEN'S CONTENTS IN AN ARRAY. HERE IT IS:

```
       ∇ΔGETSCR[⎕]∇
[ 0]     SCR ← ΔGETSCR ;ROW
[ 1]    ⍝ SCREEN GET, BASED ON 'DUMP' FROM JIM SWIFT
[ 2]    ROW←¯2+1↑ΔCURSOR
[ 3]    SCR← (ROW,79)ρ⎕PEEK 32768+(80×0,⍳ROW)∘.+0 78
       ∇ΔCURSOR[⎕]∇
[ 0]     R ← ΔCURSOR
[ 1]    ⍝RETURNS ROW AND COL OF CURSOR (NOTE: AFTER CR)
[ 2]    R←256 256⊤⎕SYS 45188 ⍝ USES 'TGETCURS_' FROM SYSLIB
```

NOTE THAT FOR SOME STRANGE REASON, WHILE THERE ARE 80 COLUMNS ON THE SCREEN, ONLY 79 ARE DISPLAYED ON A LINE WHEN A CHARACTER MATRIX IS DISPLAYED IN IMMEDIATE MODE.

I direct your attention to Barry Bogart's article on APL83 elsewhere in this issue. Since Barry wrote the piece, IBM has announced an implmentation of APL2 for the PC and I. P. Sharp is rumored to be working on its own APL implementation for the PC as well. Furthermore, the price of APL is coming down. Clearly, APL is really beginning to take hold with micros. I sincerely hope Commodore gets the message!

Finally, 4040 owners who are interested in the EDA disk announced last time will be pleased to learn that it has been reorganized substantially and fits on one disk. (See details, last page, this issue. Ed.)

∩∩∩

6425 31*ST ST., N.W., WASHINGTON, D.C.* 20015 *U.S.A.*

∪∪∪

THEM DURN SWITCHES ARE GONE

We heard rumors the newer SuperPETs were coming out with only two switches (again), and laid hands on one a couple of weeks ago. The two switches match the first two on all other models. Switches three and four (controlling UD11 and UD12) are gone. The question: how are ROM sockets UD11 and UD12 controlled? Answer: UD12 is controlled by poke 61438, 1 --which turns it on (0 turns it off). A cold start in 6502 brings UD12 in 'off'. We don't know about UD11, having no ROM fitting that socket, and no way to check. We hope UD11 is always on in 6502, and always off when you switch to 6809. Anyone with the word: please pass it. As to using UD12: we stuffed a POWER chip in, and got POWER after poking 61438,1. A word of warning: DON'T poke 61438, 0 to turn the UD12 off until you have exited the program: for POWER, you leave with OFF <RETURN>. You may then safely turn the UD12 off with poke 61438,0. If you don't follow the sequence above, CRASH. As a test, we poked UD12 on, loaded POWER, used it, turned it off, and left 6502 without poking UD12 off. You're reading the result in 6809. No problems. It appears UD12 is automatically switched off when you go to 6809 mode. We have no hard information on how to distinguish the new models, but suspect you can tell by the single, monolithic mount for both switches. If that doesn't tell you what you have in hand, then

turn Bank 0 of the upper 64 on with poke 61436,0 and peek(9*4096) with the R/W switch in R/W position. Remember what you read on the screen. Then poke 9*4096,0 to put that value into the first byte of bank 0. Now, peek(9*4096). If you read '169', you have the newer machine. So far, we have not been able to poke into any bank a specific value and then read it with a peek, no matter what the position of the R/W switch, or what values are poked to the system and bank-select latches, and we've tried everything outlined in pp. 13-14, Systems Overview.

****

1983 APL WORLD CONFERENCE

by Barry Bogart
2505 West 15th Avenue
Vancouver, B.C. V6K 2Z1

This was held in Washington, D.C. in early April, sponsored by ACM (Association for Computing Machinery), leading 'academic' computer society and the only one interested in APL, which hosts an APL conference every year or two, normally alternating between Europe and North America (Finland for '84; Seattle for '85).

'83 attracted the typical 800 or so. Commodore was there, as was Waterloo, DEC, PRIME and some specialist hardware makers. The action was divided between micro, mini, and mainframe worlds, but I think most interest was in the micro APLs and in particular the IBM PC. Unfortunately, there isn't much to report on either the software or hardware side for SuperPET users. I organized a 'birds of a feather' session for SuperPET users, which happened to conflict with a similar IBM PC session. Walt Kutz, Commodore SuperPET product manager, attended our session and assured us that our machine would stay in production, but he said the 'Easy-' (and cheap) software packages for the Superpet were shelved. He did not have anything to say about the new machines (BX-256, etc.) possibly running the Waterloo software.

John Wilson of Waterloo, who developed the APL package, dropped into our session for a minute, but then disappeared to the IBM PC session, which was typical of their current interests. They now have a version 1.2 of APL for the PC, in which the infamous 'system-error-during-garbage-collection' error has been remedied, among other things. They also announced Version 2.1 for the PC, which offers such goodies as using all the memory available, and using more appropriate internal representation for numbers, i.e. integers take two bytes instead of five. They 'knew of no plans' to make any of these improvements available to any Commodore machines (including SPET).

The real action in APL micro circles has been the 68000-based machines for some time. Now you can get a VERY fast APL for practically any such machine, including Apple LISA, TRS Model 16, Fortune, and Corvus. The software costs more than the SuperPET ($2000+), but if you can afford that kind of hardware, the software is cheap by comparison. If you want to join the herds and buy an IBM PC for APL, STSC Incorporated offers a very nice and FAST interpreter for it. The latest version of Waterloo APL for it might be as nice, but I doubt it would be as fast. And if you want a REALLY fast APL on a micro (if that's the word for it), Analogic Corp. has developed a specialized array processor named the 'APL Machine', which is faster than an IBM 3091 mainframe for some calculations!

Some perspective: In 1981, SuperPET had just been announced; it offered the first really full APL implementation on a micro, and it was usually faster than competitors. Now, in 1983, the SuperPET is by far the slowest APL micro around, and the difference is clearly the Waterloo software. All of the new micro APL's have more features than Waterloo 1.1, are faster, and allow a larger workspace. But Waterloo is responding to the competition. Now, if Commodore will do the same, we may yet have a chance to jump ahead. My own wish (unfounded specu-

lation): some new Commodore machine will use the Zilog Z-8100 chip; then it can run the newest APL (Dyalog), from Dyadic Systems in Britain, and zip past all those 68000's, not to mention leaving all those PC's in the dust. Come to Seattle in '85!

****

YE GODS! A FOURTH KEYBOARD! In Issue 6, we noted that when you get the APL font in any language but APL, the SPET keyboard is neither ASCII nor Waterloo APL--but a hybrid, for which we published the key assignments. Three keyboards was bad enough, but George Cordahi had to spoil our day by informing us there's a fourth. You get it in APL whenever you poke the Waterloo font. Since all of you can print this new APL-ASCII keyboard by poking ASCII in APL and sending the results to printer, we don't need to publish it. Those that need it can knock out a sample in no time. Just be warned it's there--and it's not the same as the third keyboard. Yetch.

****

CREATING AND USING SEQUENTIAL FILES IN APL Steve Zeller this issue shows a sophisticated way to do this, but for those just learning the language, here's an easy way to save disk space and get SEQ files that you can load and run in APL, courtesy of George Cordahi of Toronto. To illustrate his method, we will use a new, fast APL screen dump (which prints overstruck characters), which came in from Jim Swift of Nainamo, B.C. Load APL, and with a clear workspace (WS), enter SDUMP (for Swift's Dump). Issue the immediate mode commands on the 1st two lines, left. After the file is on disk, untie 1, and then file SDUMP with a)WSID SDUMP and)SAVE. This puts SDUMP to disk as a PRG file. Now check the directory with)LIB, and note the large number of blocks occupied by the PRG file, as compared to the SEQ file created by WRITE. But--a problem. The SEQ file won't load, either with a)LOAD or with)COPY: file type mismatch. Cordahi to the rescue: we have two ways to solve THAT problem. First, clear WS. Method 1: enter the immediate mode command at left, which reads the SEQ file we created into WS. (Be sure to UNTIE after it's in). The method requires a lot of typing; you may find Method 2 a lot easier if you store many utilities on disk and if you want them back quickly. Method 2: Clear WS and enter STORE and GET, as listed below. Enter each separately (not side by side, as we've done to save space). When entered, save them to disk with a STORE 'STORE' and STORE 'GET'. Then take a look at the directory. No suprise, but our STORE added 'S' to the names (for SEQ files), which explains why George, in WRITing SDUMP, added an 'S' to the name of the file (not to the name of the function!). Well, if you've followed us so far, CLEAR WS. Now we'll load GET and use it to load quickly into WS our SEQ files. We load it with 'GETS' ⎕TIE 1 (and next line) ⎕FX []READ 1, as we get any SEQ file. Once it's in WS, however, it 'gets' any SEQ file we've created in the way described--and QUICKLY. It makes)LOAD or)COPY seem snail-slow. Try it with GET 'SDUMP' and GET 'STORE' (Note no suffix 'S'--GET adds it). So long as GET and STORE are in WS, you can 'get' and 'store' any number of single, utility functions quickly and easily, as short SEQ files. The method won't work for a WS holding a lot of functions, but for singleton utilities it's a gem. If, like ye ed, you're learning the language and want to keep things simple, it's a good way to start. Steve Zeller shows, this issue, how to do it with interlaced functions working together.

```
'SDUMPS' ⎕CREATE 1
(⎕CR 'SDUMP') ⎕WRITE 1
 (leave the 'S' on SDUMPS)
⎕UNTIE 1
```

```
'SDUMPS' ⎕TIE 1
⎕FX ⎕READ 1
```

```
    ∇GET[⎕]∇
[ 0] GET FILE
```

```
    ∇STORE[⎕]∇
[ 0] STORE FN
```

```
[ 1] (FILE,'S') ⎕TIE 3
[ 2] ⎕FX ⎕READ 3
[ 3] ⎕UNTIE 3
```

```
[ 1] (FN,'S') ⎕CREATE 1)
[ 2] (⎕CR FN) ⎕WRITE 1)
[ 3] ⎕UNTIE 1
```

```
      ∇SDUMP[⎕]∇
[  0]    SDUMP ;ROW;⎕IO;SCR
[  1]   'IEEE4' ⎕CREATE 1 +⎕IO←0
[  2]   ROW←¯1+1↑(256 256⊤⎕SYS 45188)-1
[  3]   SCR← ⎕XR,((ROW,80)⍴⎕PEEK 32768+(80×⍳ROW)∘.+0 79),⎕TC[6]
[  4]   SCR ⎕PUT 1
[  5]   ⎕UNTIE 1
```

Jim Swift's dump is not the one-liner we expected, but it is short; lest readers think us dump crazy, we printed this to show how ⎕ SYS is used in APL, and to demonstrate another way to use the system library. Jim's call, at SYS, is to the decimal address of TGETCURS_; he senses cursor position on the line of the call to SDUMP, and stops the dump on that line (we modified it to stop on the line above the call). Note how Jim uses APL to convert the return from TGETCURS_ to row and column (more on this in next issue). We add that the dump is darned fast, and fully automatic--it dumps everything down to the cursor. And we had a better reason to print it: Converted, this dump will send to disk as a TEXT SEQ file any APL material on screen--WITH line numbers and fully overstruck characters.

If you send that file to any letter-quality printer which handles APL in XR, you output at printer EXACTLY what was on screen. (If that sounds simple, you've not lived until you've had to do it by converting APL functions to character matrices and then filing those to disk as TEXT SEQ files--a horrid process). Anyway, for those who can use it, herewith the converted function DTODISK; call it with the filename for the disk file, as in: DTODISK 'GEM' <return>. It creates a file named 'GEM.TXT'. Two warnings. Don't call these files into the mED, revise them, and then refile to disk, or overstruck characters disappear at printout. Second, expect to see overstrucks as SEPARATE characters if the file is reviewed in the mED, and don't print them from the mED. Same problem. Print to printer only with a printfile program, from disk, in whatever language. That works splendidly.

A last caution: every overstruck character explodes into three characters when put to file; if your text line is 80 characters long WITH an overstruck, you've tried to cram 82 characters into 80 columns. So shorten long screen lines by two characters for every overstruck character in that line--or thou art truncated!

```
∇DTODISK[⎕]∇
[  0]    DTODISK NAME ;ROW;⎕IO;SCR
[  1]    FILE←'(T)',NAME,'.TXT'
[  2]   FILE ⎕CREATE 1 +⎕IO←0
[  3]   ROW←¯1+1↑(256 256⊤⎕SYS 45188)-1
[  4]   SCR← ⎕XR,((ROW,80)⍴⎕PEEK 32768+(80×⍳ROW)∘.+0 79),⎕TC[6]
[  5]   SCR ⎕PUT 1
[  6]   ⎕UNTIE 1
```

****

1200 RECORDS IN A RELATIVE FILE?
YES, INDEED

After reading Loch Rose's article on relative files, last issue, Frank Brewster of Bradford, PA took the tip that some 8050 drives will accept more records. On his Micropolis Drive (the one with the doors

that go 'down' to close and 'down' again to regurgitate a disk--sometimes), he found he could enter 1200 records. In his words: '...how much mickeymouse I've gone through, assuming that I mustn't exceed 720 records. Send thanks to Loch ... also you can, contrary to the manual, input less than all of the data elements on one record without error messages. I mentioned this to Waterloo, who said this was a lucky accident...they had meant to allow fragmentary reads only via the ';', so the statement 'input #3, rec=120,a,b' will work even if 'b' is not the last data item on the record. And thank God!'.

* * *

THE MCFARLENE APL DISK (aka the Aussie Jewel)

After the last issue went to press, Steve Zeller sent us a copy of his revision of the McFarlene EDA disk announced last issue. The original was very good, but the reorganized version is even easier to use (and takes up a third of the disk space), so we can get it on one (not three) 4040 disks. Instructions are on the disk. Those who ordered 4040's will receive a rebate of $10.00. Either 4040 or 8050 formats are available. In 8050, write Steve Zeller (address in his column, this issue); in 4040, the Secretary (address, application form, last page). Price for all formats: $10.00 U.S. If you enjoy numerical detective work, this is a powerhouse. Steve did a superb job in reorganizing it. Even the editor can figure it out....

****

THE CUPBOARD IS BARE!!!

Byte Magazine will arrive next month whether or not you contribute material. But the Gazette will not. Want twenty blank pages next issue? Blank issue guaranteed if you sit on your behinder and assume somebody else will write. CONTRIBUTE!!!!!

Send contributions in either 4040 or 8050 format to the editor, SuperPET Gazette at PO Box 411, Hatteras, N.C. 27943. Hard copy is nice but not required. Please stick to one subject and explain what's on disk. Don't send a disk full of your favorite programs and expect ye ed to read your mind. Disks will be returned to you within three days. Short notes and articles especially wanted.

??

BITS,BYTES ERROR THIS ISSUE

The programmer is responsible for pulling the value of parameters passed to library routines off the stack. Include the following code line: leas 2,s. And note Gary told you....

??

Prices, back copies, Vol. 1 (Postpaid), $ U.S.

No. 1: not available	No. 4: $1.25	No. 7: $2.50
No. 2: $1.25	No. 5: $1.25	No. 8: $2.50
No. 3: $1.25	No. 6: $3.75	

Send check to the Editor, PO Box 411, Hatteras, N.C. 27943. Add 30% to prices above to cover additional postage if outside North America.

****

DUES IN U.S. $$ DOLLARS U.S. $$ U.S. $$ DOLLARS U.S. $$ U.S. DOLLARS $$

APPLICATION FOR MEMBERSHIP, INTERNATIONAL SUPERPET USERS' GROUP

Name:_____ Disk Drive: _____ Printer:_____

Address:_____

Street, PO Box City or Town State/Province/Country Postal ID#

For Canada and the U.S.: Enclose Annual Dues of $15:00 (U.S.) by check or money order, payable to SPUG. DUES ELSEWHERE: $25.00 U.S. Mail to: Paul V. Skipski, Secretary, SPUG, 4782 Boston Post Road, Pelham, N.Y. 10803, USA.

Newsletter published by the International SuperPET Users Group (ISPUG); a non-profit association; purpose, interchange of useful data. Editorial offices at PO Box 411, Hatteras, N.C. 27943. Secretary, Paul V. Skipski, 4782 Boston Post Road, Pelham, N.Y. 10803. Membership applications, dues, and inquiries to Mr. Skipski; newsletter material to Hatteras, attn: Dick Barnes, Editor. SuperPET is a trademark of Commodore Business Machines, Inc.; WordPro a trademark of Professional Software, Inc. Enclose a self-addressed, postpaid envelope with all material submitted and all inquiries requiring reply. Membership: $15.00 per yr. U.S. in North America, $25.00 overseas and elsewhere. See enclosed application.

For all outside the U.S.: All nations members of the Postal Union offer certificates good in the postage of any other country for a small charge. The Union includes most nations of the world. Canadian members: send Canadian dimes or quarters for postage, but no paper currency.

SuperPET Gazette
PO Box 411
Hatteras, N.C. 27943
U.S.A.

PRINTED MATTER