SUPERPET GAZETTE

TRANSCONTINENTAL TELECOMS AND GOOD NEWS AT 1200 BAUD

John Frost of Seattle, who wrote our last article on telecommunications, will pay Ma Bell's next dividend. He offered ye ed the privilege of making the first logged transcontinental SPET to SPET TC, Hatteras to Seattle, and we took him up on it. Ye ed was in TALK; John used his package for 6502. We had no trouble, and uploaded a good-sized file to John. A few nights later, we loaded Jeff Larson's FASTERM (a 6809 machine-language program that can run up to 1200 baud) and had a good half-hour on line with Jeff; his FASTERM is a jewel, giving both operators a full screen (unlike the single line in 'talk'), and giving both a capability to upload and download. Which we both did successfully. You have full cursor control; see your programs as they upload and download on screen. Later, we ran a long conversation in FASTERM with both John Frost and with Terry Peterson in California, up and downloading with ease. FASTERM is the only software needed for TC in 6809. (Yes, 300 baud is OK.)

John Frost has tested FASTERM with Compuserve, a Boeing computer, and with local BB's, and his reports are enthusiastic. He considers it superior to the 6502 TC package, which has up to now been the only game in town. FASTERM uses the standard RS232-modem cable (with no jumpers). Jeff has provided a simple way to send any CONTROL code from FASTERM along with instructions on on how to use the program. We're glad to say that you can name the files you want to send from disk, and can assign filenames to incoming files.

FASTERM will be available on disk, 4040 and 8050 format, on October 17. The program is ready now, but we must have time to include full instructions and data. When finished, the disk will include the Larson and Frost telecom articles from the Gazette, instructions on CONTROL Codes, general instructions on telecom in SuperPET, and Larson's instructions on FASTERM. The price will be $15.00 U.S., a part of which goes to the author. You may order 4040 format from the Secretary (address on application form, last page), or either 4040 or 8050 format from the editor, PO Box 411, Hatteras, N.C. 27943. Make checks out to ISPUG. We're mighty pleased with FASTERM, since it handles all 6809 TEXT, SEQ files, with no limit on the number of disk files sent or received. Jeff Larson reports he has transmitted files at 4800 baud okay (screen's not readable) but we haven't tested at that rate.

Shortly after, we'll have the new SUPERCOMX available (a revised 6502 TC package, with proper handling of the ACIA chip, as debugged and as revised by Terry Peterson). It's working now, but needs packaging. We're going to put both TC packages on one disk, with instructions, for $20.00. Those who want a SUPERCOMX update to any FASTERM disk ordered now can get it later for $5.00. We've held off announcing both programs until they were tested. Please don't order SUPERCOMX until we announce the disk is ready, with full instructions.

****

SEPTEMBER MEMBER, YOUR MEMBERSHIP EXPIRES WITH THIS ISSUE!

<u>You won't get a bill or another notice.</u> If the address label, this issue, reads e:9-nn-82 (where nn is any date last September), this is your last issue. Please send in yearly dues of $15.00 U.S. for memberships in North America, or $25.00 U.S. if elsewhere. And please send in the address label, so we won't have to do a special search (SuperPET is not yet smart enough to find John Q. Jones if the renewal comes in for Johnny Jones). Forward label and check to Secretary, ISPUG, made out to ISPUG, at 4782 Boston Post Road, Pelham N.Y. 10803, U.S.A. If your 'entry' date on mail label is: e: 10-nn-82 or later, you're okay.

KIND HEARTS AND GENTLE PEOPLE:
(Or How to Kill an Editor
with Goodness)

Oscar Wilde once said that those who try to do the most good often do the most evil. And in our experience, he's still dead right. We now have 23 full disks of undigested, unexplained, and undocumented programs, sent with the best of intentions by members who sincerely think they're helping by dumping on ye ed's desk forty or sixty of their favorite programs--with no explanation of 1) what the problem is, 2) how they solved it, or 3) the significance of what they did. If we started now, and did nothing else all day, we might get through them by 1986.

On the other hand, some members send in a few programs with full comment on what they did and why they did it, with instructions on how to use the programs. We can get these up and running in short order, and from them get much material for the Gazette. (Especially when the language is identified. Ever get a PRG file & wonder where in the seven languages/facilities it runs?)

We need more contributors, more articles on new subjects. Don't worry about writing well. Stuffin' in proper grammar and format is our job; we won't embarrass you by publishing bloopers. There's not a one of you who doesn't know something about SPET that nary a soul of the rest of us does. Tell us about it!

Anent disks: As of Aug. 1, all contributions should be on disk. We now have both 8050 and 4040 drives and accept disks in either format. Your disk will be mailed back within three days of receipt, after we've copied and had a preliminary look at your material. Send text files either in WordPro or as microEDITOR files in 6809; programs in whatever language you wrote them in. If you send APL, put a WS on disk separate from text, ready to run. We'll convert to proper format. If you write text in Wordcraft, use its facilities to create an ASCII file for the mED, as explained last issue. Hard copy of your contribution is not needed, though we welcome it. Now, durn it, git to writin'!

****

THE LOST IS FOUND
or
WATERLOO TO THE RESCUE
(And a Bollix or Four)

About two months ago, we wrote Waterloo Computing Systems, Ltd., about some problems in SuperPET, and are happy to report we have either solutions to the problems or suggested ways around them. Let's take the problems one at a time.... First, that perennial problem of leaving a language and remembering you didn't save to disk, or, Lord help us, encountering the 'bye' problem in mBASIC, wherein you see upon the screen <beginning of file> <end of file>, assume you are in the microEDITOR, and say 'bye', only to find you've said goodbye to mBASIC, your program, and all thy labor.

Waterloo Computing Systems, Ltd. speedily sent back to us a splendid solution, (two solutions, as a matter of fact), which operate in most SPET languages and facilities. The first is a hardware solution: put a switch to ground from NMI pin on the 6809 chip itself. If you inadvertently exit any language or facility, turn the switch ON, ground the pin, and you're back in that language, with your program. CAUTION!!!! The switch must not bounce (make more than one contact), or we are warned of consequences dire and horrid. The second solution: a software retrieve. That solution, written at Waterloo, is up and running, and we print it below. We've been using it enough to report, and it is as reliable as death and taxes. It 'retrieves' both your language (or facility) AND your otherwise dear departed program.... It works in everything except DEVELOPMENT.

```
; retrieve.asm, written by Waterloo Computing Systems Ltd.

Process equ     $2a                     ; Ed: Put the load module of this pro-
ProcALV equ     $39                     ; gram on the language disk as 'retrieve'
CurBnk_ equ     $220                    ; and call it with: retrieve <RETURN> if
BnkCtl_ equ     $effc                   ; ever you want a language and its pro-
                                        ; grams back.
retrieve equ    *
         ldd    #ProcALV ; get address of process auto-load vector
         std    Process  ; point to it
         ldb    #$00     ; select bank 0
         tfr    cc,a     ; save condition code register
         sei             ; disable interrupts
         stb    CurBnk_  ; remember new bank number
         stb    BnkCtl_  ; set new bank number
         tfr    a,cc     ; restore condition code register
         ldd    #-1      ; NMI reset is TRUE
         std    2,s      ; replace old parm value
         ldd    #1       ; INIT code
         jmp    $9000    ; leap to whoever is in Bank-Switched memory
         end

"retrieve"
org $680                ; this is retrieve.cmd, the linker file.
"retrieve.b09"
```

Don't use the program to return to DEVELOPMENT; at best, you'll get the menu of the language disk back; at worst, you'll crash. And don't use RETRIEVE to return to whatever language you were in, in 6809, after being in 6502 mode. Crash. Instead, use the method outlined below.

Waterloo points out that one can re-enter any language or facility in 6809 simply by putting the R/W switch in READ, and attempting to load any language (the microEDITOR, being short, runs fastest). The stupid loader thinks it is loading, and when it is through, says 'tell the operator I'm done and he has a program'. The program, of course, is that still resident in the upper 64. You can use the same trick when you leave 6809 for 6502 and come back.

In this connection, we got a short, sweet program called RESET either from Jim Swift or Barry Bogart, which does exactly what we describe above--get your language back after you leave the language or when you come back to 6809 from 6502. We sent a copy to Associate Editor Terry Peterson, and he promptly disassembled and rewrote it so you don't even have to put the R/W switch in READ. It won't get your program back, but it rescues languages with mercurial swiftness.

* * *

RESET0 : TO HAVE AND TO HOLD a Terry Peterson Special

Printed below is an mBASIC program which creates a load module for the language disk to perform the job we describe above. You need not flip the R/W switch to 'read', because the load module loads absolutely nothing. Give the mBASIC program away; you won't need it after the module is on disk. (And note we create a TEXT,PRG file. Terry wouldn't have gotten away with this if he hadn't suppressed the trailing CR on line 140, for it would have been in the file.)

```
100 ! reset0:bu : by Terry Peterson
120
```

SUPERPET USERS!

%%%°%%%&%%%.%%%$%%%°%%%&%%%.%%%$%%%°%%%&%%%.%%%$%%%°%%%.%%%&%%%$%%%°%%%.%%%&

Have you ever wondered what the /%*%$%& is the difference between 'c*/ %*//' and '*c/ %*//' ? Tired of flipping that switch just to do a 'collect'? The SuperPET Tutorial Disk reveals the mysteries of the data editing commands and 'meta-character' strings, using clear and useful examples. It also contains:

A GENERAL NARRATIVE DESCRIPTION OF THE MICROEDITOR

- ° SYNTAX AND EXAMPLES FOR ALL MICROEDITOR SEARCH STRINGS.
  EXAMPLES AND EXPLANATIONS OF ALL MICROEDITOR COMMANDS.
  EXAMPLES AND EXPLANATIONS OF ALL MICROMONITOR COMMANDS.
  EXPLANATIONS OF ALL SETUP MENU OPTIONS.
  COMPLETE INFORMATION ON THE PROGRAMMED FUNCTION KEYS.

- ° INFORMATION ON ALL FILE TYPES AND FORMATS

- ° EXAMPLES OF ALL VARIATIONS OF THE DISK ACCESS COMMANDS.

- ° INSTRUCTIONS ON ISSUING ALL DOS COMMANDS FROM THE EDITOR.
  EXPLANATIONS OF ALL DOS ERROR MESSAGES.
  INSTRUCTIONS ON AUTOMATING DISK MAINTENANCE TASKS.

- ° INFORMATION ON RS-232C AND THE TERMINAL FACILITIES.

- ° A TABLE OF IMPORTANT SYSTEM ADDRESSES AND SOFTWARE SWITCHES.

- ° DECIMAL AND HEX VECTOR ADDRESSES OF WATLIB AND FPPLIB ROUTINES.
  6809 ASSEMBLER INSTRUCTION OPCODES, MODES, AND LENGTHS.
  HEXADECIMAL-DECIMAL CONVERSION TABLE.

- ° HEXADECIMAL AND DECIMAL ASCII CHARACTER TRANSLATION TABLES.

THIS PRODUCT COSTS ONLY $39.95, POSTAGE AND HANDLING INCLUDED. THE ITEMS MARKED WITH '°' ARE ALSO AVAILABLE ON A REFERENCE CARD WHICH IS INCLUDED WITH EACH TUTORIAL DISK ORDERED. THE REFERENCE CARD ALONE COSTS ONLY $10.

IF YOU ORDER ANY DISK-BASED PRODUCT, THE DISK YOU GET WILL ALSO CONTAIN A SELECTION OF THE BEST PUBLIC-DOMAIN SUPERPET SOFTWARE FROM VARIOUS SOURCES. ALSO AVAILABLE IS THE APL-MICROEDITOR INTERFACE; IT ALLOWS USE OF THE MICROEDITOR FOR EDITING APL FUNCTIONS AND VARIABLES. VOLUME DISCOUNTS ARE AVAILABLE (30 PERCENT OFF FOR 2-10; 40 PERCENT OFF FOR 11-100.

SEND A CHECK NOW (AND SPECIFY 4040 OR 8050 FORMAT); OR WRITE FOR INFORMATION TO→

DYADIC RESOURCES CORPORATION
2405 WEST 15TH AVENUE
VANCOUVER, B.C. CANADA V6K 2Z1

('c*/ %//' hangs up; '*c/ %*//' does nothing; '*c*/ %*//' removes spaces from left)

```
130 open #9, 'reset0,prg', output
140 print #9, chr$(hex('90'));rpt$(chr$(0),3);chr$(1);rpt$(chr$(0),5);
150 print #9, chr$(2);chr$(0);
160 close #9  ! Terry lists the code created and annotates it as follows:
```

byte	header					trailer					
count:	1	2	3	4	5	6	1	2	3	4	5	6
(hex)	90	00	00	00	01	00	00	00	00	00	02	00
	load address		block length		switch to	bank 0		ignored			eof flag	ignored

He comments that "the file consists of a 'header' 6-byte block for a zero-length load record plus an end-of-file 6-byte trailer, which is normal machine language 'PRG' WCS file format, except, of course, that the load records are normally >0 in length. It is fortuitous that a zero-length record may be handled without a system crash. The code causes the WCS loader to set the relevant system pointers to begin execution of a program in bank 0, address $9000. Then 0 bytes are loaded into RAM; the eof trailer is encountered, so the load halts and execution begins. The loader 'thinks' it is starting the file it just 'loaded', when in fact the already-resident interpreter is all that gets turned on!"

* * *

...a flea hath smaller fleas
that on him prey, and these
have smaller still to bite 'em
and so proceed, ad infinitum

With apologies to J. Swift, we note that computer programs have bugs, which are bedeviled by glitches, which in turn are afflicted with a bollix or two. Herewith a report on a glitch and a bollix or so in SuperPET. The first is a large glitch, as glitches go. Load the microEDITOR, and pull any text into it which is longer than one screen page. Cursor down to <end of file>, hit PF0 (or SHIFT 0 on keypad), and enter one blank line at end-of-text. Screen cursor must be ON <end of file>. Then cursor up until <end of file> disappears below screen bottom. Cursor back. Caught a glitch! Note that a screen line from the file has been duplicated just above <end of file>. This glitch is repeatable. So long as (1) screen cursor is on <end of file>, (2) you insert a blank line with PF0, and (3) you enter no text on that new, blank line, the mED copies a line from previous text to the new, blank line. You cannot see it on screen unless you run the new, blank line off-screen and then cursor back. Solutions: (1) use insert mode when working above <end of file>, or (2) hit ESCAPE after entering a new, blank line with PF0 at <end of file>, or (3) put text on that line and <RETURN>.

Waterloo reports the glitch has been squashed in the software (Version 1.2?).

* * *

THE DEADLY BOLLIX

With something (whatever) on screen in the microEDITOR, send a command to either file it to disk or to printer. Then print: ? <RETURN> at command cursor, to duplicate your file command. We are about to catch a bollix, so be careful. With the retrieved file command on the screen, touch UP/DOWN CURSOR. Aha! Your command is executed; bollix caught. If you care to, try the same thing with the RUN/STOP, OFF/RVS, CLEAR/HOME, or any PF key. Bollix again. Any of those keys will execute the command retrieved with ? <RETURN>. Queried, Waterloo confirmed the bollix and said they'd try to find a solution. Meanwhile, you are well advised to be extremely careful of what keys you touch after retrieving a command in mED with ? <RETURN>.

****

TO CHANGE DEVICE NUMBER ON DISK DRIVES

Associate Editor Terry Peterson just put a new 8250 to work and must have kept his old 4040, because he sent in the following program to change device number; we

```
 5 ! change_device.bu
10 ! This program sends the appropriate 'pokes' to a
20 ! 4040/8050/8250 (& hard disks, I think) to cause
30 ! a soft reset of its ieee-488 bus address.
40
50 input "What is old address? ",old_address%
60 open #10,'ieee'+value$(old_address%)+'-15',output
70 input "What is new address? ",new_address%
80 new_lsn% = new_address%+32 : new_tlk% = new_address%+64
90 print #10,'M-W';chr$(12);chr$(0);chr$(2);chr$(new_lsn%);chr$(new_tlk%)
100 print "OK - device is now ";new_address%
110 close #10
```

just got a rebuilt 4040 up and running so we can handle all disk formats, and used it to set the 8050 and the 4040. Both drives wagged tails and did our bidding as devices 8 and 9.

****

A BUG IN THE 4022? Don C. Momberg sent a note after reading about the problems described last issue in getting brackets '[]' to print properly when converting BASIC 4.0 cursor commands to English with LIST-9. Seems a bug infests the 4022 (and maybe other Commodore printers), which Don learned of while reviewing the PaperMate wordprocessing manual (Michael Riley, V2.0, issued by AB Computers). (We're not so sure it's a printer problem: see below.)

If the 4022 is in lower-case mode, left and right brackets, up-arrow, back arrow (underline) and backslash all print as graphics characters. Likewise, when the substituted graphics characters are used in text, they print the characters we list above to printer.

Don sent a BASIC 4.0 program to demonstrate, but we can't print graphics on our printer. You'll have to run your own demo. But we note that the 'cross' and the 'bar' we noted as a bug in the LIST-9 conversion process (substituted for '[]') appear on our screen as well as at printer in BASIC 4.0. So is the bug really in the 4022? Don notes also that the bug does not appear in printouts from the mED in 6809 (or as we've noted, in any 6809 language). Nice job by Waterloo. Suspect a glitch in the 4.0 ROMs.

****

TWO KEYBOARDS FOREVER Or, the Editor is Blind! Sometimes an editor doesn't do his homework. We try. But... A couple of issues back, we complained about three keyboards in SPET, and last issue, about there being a Fourth Keyboard, obtained when you poke the Waterloo Roman font while in APL. Shortly after, Dr. John C. Wilson of Waterloo dropped us a note:

"I've noticed a bit of confusion about the character sets on the SuperPET....Use a SYS (to 45194) to change between ASCII and APL characteer sets. If you POKE to change fonts, only the screen driving hardware is affected; the keyboard/screen managing software is not aware of the change and this leads to rather peculiar behaviour. When you SYS to the font-changing routine at 45194, the hardware change is made for you, and in addition the software learns about it. The argument of the SYS is 1 for ASCII and 2 for APL."

Before we continue: 45194 decimal is hex $b08a. When we got Dr. Wilson's letter, we took a quick look at watlib.exp on the language disk, and there the routine was: TSetChar_, at $b08a. For a year it's been there.... From now on, there are only two keyboards in SuperPET.

Since there is no SYS capability in the microEDITOR, and mBASIC requires a program to pass parameters, we came up with a simple, standard way to get either of

of the fonts and the proper keyboard from the monitor. Enter it from menu on the language disk or from the microEDITOR, in whatever language, and at the prompt, >, enter the code below, modified from that in Barry Bogart's article in Vol. 1, at pp. 25-26.

```
>m 7000 cc 00 02 ed 8a bd b0 8a 3f  [command for APL font and APL keyboard]
```

Hit <RETURN> on the line above, and then say, at prompt: >g 7000. You will be in the APL font immediately with the standard APL keyboard. Then quit the monitor, at prompt > with q <RETURN>. When you want ASCII (the Waterloo Roman font) back, issue the same command in the monitor, with a single change: following cc, enter 00 01 (recognize we're passing the font/keyboard argument of 1 or 2 at this position). Anyone who cares to can do a direct SYS in any language but mBASIC.

****

'NOT TO_BE' or 'NOT TO_BE%'? Yes, That IS a Question!

by T.M. Peterson

[Ed. In a letter to Associate Editor Terry Peterson, we noted that 'if not idx...' fails to determine if idx has value; idx is an intrinsic function in mBASIC, fast and powerful, which finds a string within a string. Turns out it returns an integer value, and....]

The evaluation of 'logical' expressions in mBASIC is rather subtle if one is not dealing with 'logical' quantities. The root of the 'problem' is that mBASIC treats everything except integer quantities in a consistent fashion, namely, zero equals 'false' and non-zero equals 'true'; and 'not' toggles the two. Therefore, if one evaluates 'not sin(.3)' the result is '0', i.e., false, since sin(.3) is non-zero or 'true'. Similarly, the result of '2 and 8' is '1'. You can verify this by PRINTing it.

However, in the case of integers mBASIC performs bit-wise operations on the operands instead of simply testing whether they are zero or not. In particular, 'not' makes every bit of the result the 'opposite' of the corresponding bit of its argument, so '0' becomes '-1' and '1' becomes '-2'(!). Therefore, if we set x% to 2 and y% to 8 the result of 'x% and y%' is '0'--and 'not x%' is '-3'! At first, this may seem inconsistent--and in a sense it is!--but this treatment of integers allows more flexibility than one would have if integers were not singled out for special handling. For example, to print reverse-field characters on the SPET CRT you need to set bit 7 before printing. This may be easily accomplished via 'print chr$(b7% or ord(c$))' where b7%=128 and c$ is the character you want 'reversed'. If integers (yes, 'ord()' is an integer) weren't handled differently from floating point quantities, you would always get chr$(1) with the above! Now, before you interrupt, I hasten to add: Yes, in the example I just gave you could have used 'chr$(128 + ord(c$))', but suppose the routine you put this in were passed (inadvertently or not) an already-reversed character (ord(c$)>128). In this case, '+' will cause an error while 'or' will print the reversed character!

In general, mBASIC's treatment of logical expressions will behave as you expect (zero=false, non-zero=true), as long as you avoid integers except when you WANT bit-wise operations. However, you must be careful when dealing with string functions because those string functions that return numeric values return INTEGER numeric values. See the example at left. It will (almost) never execute the 'else' clause.

```
100    if not idx(a$,b$)
110      ......
120    else
130      ......
140    endif
```

****

SUPERPET FILES : DISK FILES
A CONCEPTUAL FRAMEWORK

For nineteen months, the editor has been puzzled by lack of a coherent explanation of the types and formats of SuperPET files. We've asked a lot of people to explain the file structure and until recently got no answers. As Associate Editor Terry Peterson wrote:

"The Waterloo documentation is rather sketchy about the various disk files that the SPET is capable of using. In fact, I'm reminded of Prof. Peter Schickele's analysis of the plot of 'Hansel and Gretel and Ted and Alice' (P. D. Q. Bach's opera in one unnatural act): At first, it appears complicated, but--with a little study--it seems hopelessly confused. However, I'm being a little unfair. The documentation is mostly correct--as far as it goes...."

What are the problems, the quandaries, the unanswered questions?

The System Overview and Language Manuals describe TYPES and FORMATS of SuperPET files as shown left, below. Can you explain why the disk filetypes PRG and USR nowhere appear on the list? What is the relation between SEQ, PRG, USR, and REL designations on disk directories to the file types and formats show at left? Do you know? Next, note that we can combine the Waterloo file TYPES and FORMATS in the matrix at left. Have you ever seen a Variable, Seq file on directory? How about a Fixed, Seq? And how many of the combinations shown are useful? For what? Do you know? If not, read on, for it has taken some four months for ye ed, Associate Editor Terry Peterson, and for P.J. Rovero to piece this puzzle out. We suspect Terry is the retired Sherlock Holmes in disguise, because he certainly did a superb piece of detection as this matter proceeded, and is largely responsible for what we have learned. First, as he pointed out, there is a relation between Waterloo's filetypes of Sequential and Relative and the DOS formats of SEQ, PRG, REL, and USR, but that relationship is at first most obscure.

File TYPES: Text, Fixed, Variable

FORMATS: Sequential, Relative

Text, Seq
Text, Rel
Fixed, Seq
Fixed, Rel
Variable, Seq
Variable, Rel

Second, the Waterloo formats do not seem to be detailed formats, but merely an indication of the ORDER in which data are stored and can be retrieved. To make the distinction clear, let's call the Waterloo designations ORDERS. Once we make this conceptual leap, the structure below is clear. The Waterloo ORDERS have nothing to do with what appears on disk directory as filetype, except in the case of Relative (Waterloo) REL (DOS) files.

Waterloo file TYPES: Text, Fixed, Variable

Waterloo file ORDERS: Sequential, Relative

DOS 'sequential' FORMATS: SEQ, PRG, USR

DOS 'relative' FORMAT: REL

Well, with the concept in hand, we list below all the possible combinations of Waterloo and DOS orders and formats, and print Waterloo ORDERS in lower case to distinguish them from the DOS formats in upper case.

Possible File Types/Orders/Formats:		Applications:
1	Text, seqorder, SEQFORMAT	Default in all but APL.
2	Text, seqorder, PRGFORMAT	APL PRG files; 'stored' mBASIC files

```
3       Text, seqorder, USRFORMAT          Not explored.
4       Text, relorder, RELFORMAT          It really exists!

5       Fixed, seqorder, SEQFORMAT         Likewise exists.
6       Fixed, seqorder, PRGFORMAT         Likewise exists.
7       Fixed, seqorder, USRFORMAT         Likewise exists.
8       Fixed, relorder, RELFORMAT         The normal relative file.

9       Variable, seqorder, SEQFORMAT      Not explored.
10      Variable, seqorder, PRGFORMAT      The usual '.mod' file in Assembler
11      Variable, seqorder, USRFORMAT      Not explored.
12      Variable, relorder, RELFORMAT      Not explored.
```

Since the Waterloo ORDER 'sequential' is a default in all filetypes but DOS REL, you may safely and utterly neglect it. And, since the DOS format REL is created whenever you file a 'relative' Waterloo file, you may also ignore the duplicate structure set out above for REL files. But the table makes the file structure in SuperPET crystal-clear.

```
1000 ! text seq seq file 'text.seq.seq'
1005
1010 scratch 'textfile'
1020 open #6,'(text)textfile,seq', output
1030 print #6,'A'
1040 print #6,'AB'
1050 print #6,'ABC'
1060 close #6
1090
1100 on eof ignore
1120 open #2, '(text)textfile,seq', input
1130 for i=1 to 10
1140   linput #2, aline$
1150   if io_status <> 0 then quit
1160   print i,"'";aline$;"'",'length: ';len(aline$)
1170 next i
1180 close #2
```

Terry Peterson sent us a whole slew of test programs, one of which we copy at left. Modify lines 1020 and 1120 for file combinations wanted. Try '(text)textfile,prg', and you will get a file in sequential ORDER and in DOS format PRG. Then manipulate the program through the other and sundry combinations, as we did. (microBASIC does not support variable files.)

Having established the framework, let's pass on to Terry Peterson's article. He does a superb job in defining and explaining the structure and uses of various file combinations. After that, we briefly discuss how to structure filenames in SuperPET.

* * *

FILETYPES, SCHMYLTYPES: What's Going on, Anyway?
by T.M. Peterson

First, let's clear up a bit of terminology. In the various documents that accompany your SPET and CBM disk drives there are two essentially different references to disk 'filetype'. These two kinds of 'file type' are specifiied in the 'file-designator' part of the 'filename' in all the Waterloo languages, one in front of the ',', and the other afterward, as shown at left, below. The one preceding the commas [as in (text)], I will refer to as the Waterloo filetype, and the other I will call the file organization, as in 'seq' or 'prg' (we can also call it a CBM 'filetype'). [Ed. We've called it disk FORMAT in the article above.] If you examine the Waterloo manuals, note that when they're really careful about it e.g., as in the COBOL manual [Omigawd, you didn't read that thing, did you?]--the Waterloo folks use this same terminology. The CBM literature refers to two flavors of CBM 'filetype' (or file organization): sequential and relative.

```
'(text)example,seq'  or '(t)example,seq'
'(fixed)example,rel' or '(f)example,rel'
'(var)example,prg'   or '(v)example,prg'
```

Unfortunately, the the CBM 'sequential' filetype has three different names: SEQ, PRG and USR. The contents of these three kinds of sequential files are indistinguishable as far as the DOS is concerned. (I should here define DOS: it is the program executing within the CBM drives 4040/8050/????, as opposed to the file-handling part of the operating system in SuperPET.)

All of the sequential files consist of linked lists of arbitrary bytes--accessible one after another. Of course, since these files are conventionally used for different purposes, you must request the 'correct' sequential type (SEQ, PRG, or USR) in order to get the DOS to allow you to access the data within one of these files. Yet the only real distinction between them is the setting of particular bits in the file directory on diskette. All of them are Sequential files, in the sense that the data are filed and retrieved sequentially.

If you wanted to access a particular PRG file as a SEQ file, you would need only to change the file directory bits so that the PRG file becomes a SEQ file on the directory (without any change of contents). If, for example, you 'store' an mBASIC program to disk as a PRG file, and then toggle the directory bits to make it a SEQ file, you may try to 'old' it (as a SEQ file) back into memory. At this point, the DOS will allow you to 'open' the file and will attempt--without succes--to carry out the load. The load fails because mBASIC sends quite different byte streams to disk when 'saving' and 'storing'. 'Save' sends an ASCII translation to a file; 'store' actually copies memory contents to file. (All the interpreters, save those for mBASIC [store] and APL, save an ASCII listing to disk.)

On the other hand, the DOS maintains REL files quite differently. Any REL file is actually two files (1) a collection of fixed-length records containing the data, and (2) a linked list of pointers to the data records. This dual file organization allows one to access the data on REL files in an arbitrary order.

As far as the Waterloo software is concerned, the choice of SEQ, PRG, or USR is arbitrary, except that 'SEQ' is assumed if no other file organization is specified. Within the Waterloo languages, the REL file organization (CBM filetype) should only be used with the Waterloo 'fixed' filetype, as explained below. Let me also point out that the 'direct access' files referred to by CBM in its disk drive documentation are not to be confused with REL files. Direct access files are not actually 'disk files' at all, but merely a method of bypassing the DOS file structure to access disk blocks directly by track and sector number.

Having disposed of the file organization or 'CBM filetype', we are left with the chore of filling in the gaps in the Waterloo documentation concerning the Waterloo 'filetype'. As mentioned in the System Overview Manual, there are three 'filetypes': 'text', 'fixed', and 'variable'. Text files are the type you're probably used to if you're an mBASIC programmer. These files consist of a series of 'records' delimited by carriage returns. That is, the actual file contents are a series of bytes interspersed with chr$(13)'s that are interpreted as record separators. Further, commas within each record may delimit intra-record fields, as in the example at left. Records in text files are of variable length up to some system-dependent maximum; in BASIC 4.0, the maximum is 80 characters; in microBASIC, it is effectively 'free memory."

```
THE FIRST RESULT, 123.456 <CR>
```

'Fixed' files contain records all of the same length. Unlike text files, there is no record delimiter written into the file, since it is unnecessary. (The Waterloo file-handling system keeps track of record boundaries.) Fixed files may

be written in either sequential or relative organization. If they are 'relative', individual records may be accessed at random using a 'record number' specification in the language-dependent input/output statement.

WARNING! If you use fixed files with sequential organization, the DOS maintains no information on the file about the record length (the DOS does not know that is a 'fixed' file); you may specify a different record length when opening the file to 'read' it than when you wrote the file--with perhaps disastrous results. This error is not possible with relative organization because the DOS will not allow you to 'open' a REL file with an incorrect record length specification.

'Variable' files attempt to combine the flexibility of the arbitrary record length of text files with the freedom from contextual interference of 'fixed' files; there is no special 'delimiter' character, so that any character may appear within a record. This is accomplished by encoding within the file the length of each record 'outside' of the actual data. In the case of the Waterloo SPET file system, each record is preceded by a two-byte 'byte count' of the record. These files may be used in microFORTRAN and (I think) in microPASCAL and microCOBOL. I'm not sure of the situation in APL. Variable files created e.g. in microFORTRAN may be read by microBASIC (if opened as a variable type), although the mBASIC documentation claims they are 'not supported.' If you have understood me so far, you should now be able to 'trick' mBASIC into creating a 'variable' file! (An exercise for the reader.)

So, to summarize, we have three (Waterloo) filetypes: text, fixed, and variable; and two file organizations (CBM filetypes): SEQuential (which is also called 'PRG' and 'USR') and RELative. The table below shows which of these may be mixed and matched (* = OK). Note the only forbidden combinations involve REL file organizations with non-fixed record lengths. Also note I didn't say you couldn't create such files! But if you do, may the Lord have mercy upon your Soul! Think about it....

Waterloo Filetypes:	CBM Filetypes: Sequential SEQ	Order PRG	Order USR	Rel Order REL
text	*	*	*	
fixed	*	*	*	*
variable	*	*	*	

Printed at the beginning of this series is a single program in microBASIC, which is ISPUG's lingua franca, which you can modify to test the output from each file type combination. You can easily modify it to handle REL files. I learned a lot about the construction and use of SuperPET files from running such variations, and suggest you could do worse than spend a few hours learning what each file structure produces as file output in your favorite language. You are due for a few suprises, many of them pleasant. <End Peterson>

* * *

FILE COMMAND STRUCTURE: WITH EXAMPLES

The disk file commands in the System Overview manual, pp 28 ff, being generalized, fragmented, and without quotation marks where needed, are very hard to follow. We set forth below an integrated package, with quotes, with the Waterloo filetype and DOS filetype, plus some examples taken from full specification to maximum default. First, the general format of a filename:

```
          disk address - channel/drive.(Waterloo type)file-designator, DOS format
Reference       1             2      3        4              4.1            5
below:
```

Ref 1 Includes two designators: the word 'disk' plus the device number, in the form 'disk8' for device 8, 'disk9' for device 9, etc.

2 Automatically assigned by the interpreter; DO NOT SPECIFY without good reason (channels 2-14 are used when a designator is specified; channel 15 only when there is no designator. You may override auto assignment, but run serious risk of two open files having the same channel number. Channel 0 is reserved for directories and other system commands; channel 1 is reserved for system use. Neither should be opened or closed. See separate note on 'channels').

3 Drive Number: 0 or 1; default is to 0

4 Select one of three file types: text, fixed, variable. Default to text.

4.1 Commonly miscalled 'filename'. The TITLE of the file on directory. Waterloo calls it the file-designator.

5 DOS format; default is to sequential. 'rel' must be used for relative files, 'prg' for program files; 'usr' for user files.

EXAMPLES OF FILENAMES: File title is 'example 1'

For Device No. 8:

'disk8-12/0.(text)example 1,seq'	[full filename; no use of defaults; a 'text' file, 'SEQ' DOS format]
'disk/0.(t)example 1'	[same filename, some defaults]
'example 1'	[same filename, all defaults]

* * *

For Device No. 9:

'disk9-12/1.(f:80)example 1,rel'	[full filename; no defaults]
'disk9/1.(f)example 1,rel'	[same filename, all defaults; 80 bytes is default value, fixed files]

Note on Channels: DOS channels and logical file numbers are two different horses. You are not limited to numbers 2-14 when opening logical files for devices such as disk, printer, keyboard, and terminal. Logical file numbers as high as 1500 have been successfully employed. Example: open #1500, 'printer', output. We have not tested for an upper limit, 1500 LFN's being most adequate. We note this because most SPET programs we've seen limit themselves to LFN's 2-14.

READABLE CODE AND LONG VARIABLE NAMES

Being curious, we ran some tests in mBASIC and in mFORTRAN to determine how much run-time a program lost from using long variable names for both string and numeric variables (versus variable names of one character). We compared 'long_variable name_used_here' with 'a' and 'long_string_name_applied_here$' with b$--and found to our astonishment that the LONG names ran just as fast as the short ones did. So we repeated the tests with a number of short and long names in separate test

loops. Results confirmed. A 31-character-long variable name runs just as fast as a single-character variable name. Period. So, if any of you have been abbreviating or truncating 'newfile$' to 'nwfl$' under the illusion that it is faster, be informed it's not. Short, abbreviated, truncated variable names generate nothing but unreadable code. Long names may take a little longer to type, but produce a readable code that runs just as fast. We don't know how Waterloo managed it, but they managed it. Let's see no more code written 'call tmchk' when we're really calling procedure 'timecheck'. English is the ultimate programming language in SuperPET. What say we stop mumbling and speak it?

***

CHAIN DOES NOT PASS MATRIX VALUES

Despite the claim on p. 188, V1.1 mBASIC manual, that all matrices are passed to the overlay program when the keyword NAMES is used to chain (see line 190 left, below), neither identification as matrices, nor values of matrices are passed. Try the program as printed. You'll note that values of all single variables pass, as promised, to the overlay. Then make a second run, and remove the comment '!' from lines 120 & 130 of 'overlay'. You'll find the program fails to a 'not a matrix' error, even though you printed both matrices to the screen before you CHAINed.

```
100 ! 'chaintest.bd'
110 b=8 : c%=10 : d$='Frank'
115 dim f(50)
120 for i%=0 to 9
130   e(i%)=i%
140 next i%
150 mat f = (1)
160 open #3, 'keyboard', output
170 mat print e;
180 mat print f;
190 chain 'overlay', files, names

100 ! 'overlay'
110 print b, c%, d$
120 ! mat print e;
130 ! mat print f;
140 print #3, 'Test: file open?'
150 reset : stop
```

There's a second way to pass a variable to a CHAINed overlay: the USE statement. Variable names must appear in both the CHAIN (line 190) and in the overlay (new line 105, overlay), as below:

```
190 chain 'overlay', b, c%, d$
105 use b, c%, d$
```

You'll find the variables listed in the revised line will indeed pass to the new overlay. Now try passing a matrix. If you succeed, let us know how! We can't do it, and have numerous complaints that no one else can either. But there's a simple solution, provided this issue by Frank Brewster: CHAINING WITHOUT CHAINING.

***

NOTE ON THE USE OF THE KEYBOARD BUFFER

At any time a program is running, most of the keyboard's closed off to the user (but for the STOP key), but the keyboard buffer will still receive characters. Whenever the program halts for STOP, for PAUSE, or for INPUT or LINPUT statements, SPET dumps the keyboard buffer. It ALSO dumps the buffer when a program reaches its end and the system prints READY. On STOP statements in program and on printing READY, it dumps the whole buffer. On PAUSE, INPUT and LINPUT statements it dumps only to the first carriage return in the buffer. This is no suprise, since SPET looks for a statement followed by a carriage return for input, of course. Below is an example, showing the autoinput from the buffer at input, linput, pause, & READY statements. Note that 'linput #' inputs not only the answer, but also the question, as the final output string in line 140. And note also that after READY the word UNREADY prints to the screen as the buffer empties, and without quotes!

```
100 ! autoinput example 'autoinput.bd'
105 CR$=chr$(13)
```

```
110 open #20, 'keyboard', output : open #30, 'terminal', input
115 print #20, 'How much to tell me?'+ CR$ +'No!'+ CR$ +'cont'+ CR$ + UNREADY
120 input 'Want to know your future? ', ans$
125 print 'Have any money? ';  ! Remove semicolon to avoid picking up the
130 linput #30, money$         ! question as part of the answer.
135 pause
140 reset : print 'ans$ is: ';ans$, 'money$ is: '; money$
```

We suspect the keyboard buffer operates similarly in all languages. Any reports?

****

CHAINING WITHOUT CHAINING
or, How to Save those Lovely Variables with Ease
-- by Frank Brewster --

[Ed: Would you like a few megabytes of memory for SuperPET? Well, you got it. When this program first came in, we thought Frank was just playing games with the dynamic keyboard. He's done far more--he has created unlimited virtual memory for SuperPET which saves all variables--no CHAINing, no definition of variables to be saved. Frank arranges for SPET to give itself instructions to delete old, used program, to merge new program from disk, and then to continue a run. When we ran his program, we found we had more memory free AFTER the run (if we exclude memory occupied with variable values) than we had at the start. There is no doubt about the technique. It works. Better yet, it's simple. Some readers may remark that CHAINing with the 'names' statement (p.188, 1.1 mBASIC manual) is easier. Ha! See article, this issue. Can anybody make it work? We note that mPASCAL and mFORTRAN both have PAUSE and CONTINUE statements, and wonder if this technique will work in those languages also--and, perhaps, with APL, to end that terrible problem of not enough workspace.]

To have a program issue system commands to itself while it's running, you must halt it in some way, at which point the system will await instructions from the keyboard. There are several methods of halting, most of which aren't worth house room:

You can include an error; it halts program, but why bother? Or you can include an input statement (which will dump the keyboard buffer to the first carriage return okay), but the system expects data, not system commands.

Or you can include a STOP statement, which halts program all right, but the only way to get it started again is to give it a 'goto' or to call a procedure. A 'continue' command fails, as does a 'resume'; the program has lost its references. You may also STOP program with the STOP key, but how do you know where?

I've worked at it, and have found the 'pause' statement the best, in spite of the back references it prints to the screen. During a 'pause', you can enter any system commands which don't utterly wreck the program, and may then 'cont' program at will. Every 'pause' dumps the keyboard buffer thru the first carriage return in it, so if you have the right command string in the buffer, SPET will tell itself to execute that command.

Have you ever needed more memory? I've often wished the 'chain' statement would simply pass all data intact without USE statements (many of my programs have a hundred variables or more, including large matrices). Well, there's a way; as I show in the program below, you can wipe out parts of the program that have been used, bring in a new section or sections from disk, and then resume program with all data undisturbed.

I've developed a simple way to do it (it's always easy after you find out how); procedure 'command', below, is very simple and yet very powerful (hence, use it with caution). You can do most anything. Shown below is one ultimate, a program which runs until the first part is no longer needed, deletes that from memory, gets more program from disk, overlays it in the deleted area, and then continues with all data untouched.

Your only problem will be getting rid of the messy PAUSE messages on the screen. Since these messages vary in size, it's easiest to clear screen quickly.

All of the necessary operations (filling the keyboard buffer, pausing, issuing commands, resuming operation, erasing) are handled by calls to PROC COMMAND. It works this way [I use CR$ for chr$(13), and CS$ for chr$(12)]:

1. You pass c$ as a parameter to the procedure. That string is simply the system command you want stored in the keyboard buffer, including a carriage return, as in: 'del 100-250' + CR$. If you want to clear the screen and then to issue a directory command, the command string is: CS$ + 'di'+ CR$.

2. PROC COMMAND creates command$, which adds to your command a 'cont' statement and another carriage return (CR$).

3. It opens the keyboard buffer and stores the command string there. You can stuff only 40 characters in the buffer, so keep commands short.

4. It then, by pausing, causes a dump of the keyboard buffer thru the first CR$, thus entering your stored command.

5. After the command has been obeyed, the system answers 'Ready', then looks for another command in the keyboard buffer. Your 'cont' and its CR$ are already there and waiting. 'cont' is printed to the screen, the CR$ follows, and your program begins to run again.

Be sure you put the add-on program 'addlines' on disk 0 as a separate file. The program will look for it there as a SEQ (SAVED) file after it deletes the first part of itself. And don't try to RUN the program in main memory again: parts of it aren't there any more!

```
100 ! program  virtual.bd
110 CR$=chr$(13) :  CS$=chr$(12) : T$=chr$(9) : D$=chr$(10)
120 print CS$; : option base 1 : dim zilch(50)
130 for i=1 to 50
140   zilch(i)=i
150 next i
160 print 'Here is data before we delete part of program in SWITCHER.'
170 mat print zilch;
180 print : input 'When ready to proceed, press RETURN: ', o$
190 call  switcher
200 call  addlines
210 print D$;T$;T$;'End of Run. Now List the program...'
220 stop
230
240 proc  switcher
250   print CS$;
```

```
260   call  command('del 100-180'+CR$)
270   call  command('merge "addlines"'+CR$)
300 endproc
310
320 proc  command(c$)
330   command$ = c$+'cont'+CR$
340   open #3, 'keyboard', output
350   print #3, command$
360   close #3
370   print CS$;'This is command to be executed: ';c$
380   call delay(400)              ! These delays may be eliminated. They are
390   pause                        ! inserted to show command execution.
395   call delay(100) : print CS$;
400 endproc
410
420 proc delay(period)             ! You may delete this procedure, used only
430   for j=1 to period            ! for demonstration.
440   next j
450 endproc

100 proc addlines !   save on disk/0 file "addlines"
110   print 'Old Data are Undisturbed: ';D$
120   mat print zilch;
130 endproc
```

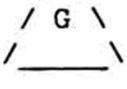
{Note that Frank merges 'addlines' at the start of the old file. He could have merged it at the end. You can merge anywhere so long as line numbers stay in order. Also note that a few modifications of 'switcher' could easily merge addlines.1, addlines.2...to addlines.n, one on each succeeding pass. Ed.]

****

A NEW HOME FOR WORDPRO AND POWER AND OTHER ROMS/PROMS/EPROMS

Last issue we noted that the latest SPETS are equipped with two switches and we listed some problems trying to load and to peek the upper 64 from 6502 mode in that model. Our dealer pleading ignorance (Customer Support at Commodore was at par), we asked Waterloo for help, and received it in time to test before this issue appeared. It seems that on two-board SuperPETs with two switches, the locations of sockets have changed. U45 now takes ROMs, etc. designed for $9000-$9FFF, and U46 those for $A000-$AFFF. The new sockets are easy to find and use, being on the front edge of the top board, and are clearly marked. As soon as we moved POWER to its new location at U45, all problems reported last issue disappeared.

Waterloo says that "for the Universal Board in the 8032 after March, 1981, no change is required. (This is the lower of the two boards.) If your SuperPET has three boards, this technical memo does not apply. On the COMBO Board [The upper of the two boards. Ed.] the following changes are required for Revision B Boards or earlier. The revision mark (a letter enclosed within a triangle) is found on the COMBO board. Revision C or later boards have had this modification made." We found our revision mark top, left, rear of the top board, about the size shown at left. Waterloo continues:



"1. Add a trace from pin 24 to pin 21 on U46.

"2. Cut the trace from pin 21 to the bottom of the board.

"Note that WordPro and VisiCalc will not work without this modification. The same applies to the Waterloo Structured BASIC enhancement chip. The WordPro chip plugs into U46 which is address range $A000-$AFFF. The VisiCalc chip plugs into U45 which is $9000-$9FFF. This address range is also the bank-switched memory range.

"To access the $9000 ROM (U45), you must (using BASIC 4.0) poke 61438,1; to access the $9000 RAM from 6502, you must (using BASIC 4.0) poke 61438,0. It is recommended that the $A000 socket be used for the Waterloo Structured Basic enhancement chip." (Emphasis: Revision C and later boards need no modification. Ed.]

We add that U45 boots up OFF. You must poke 61438,1 to turn on any ROM chip in that socket. And, as we warned last issue, you must turn off whatever program is in that ROM chip before you switch off the socket with 61438,0--or CRASH. Those of you who, like ye ed, stuffed ROMs in UD 11 or UD12 may have found that they worked, but all manner of mad things then transpired. Move 'em to U45 or U46. We again appreciate the splendid support from Waterloo, but are compelled to wonder aloud if Commodore ever will tell owners about configuration changes and how to handle them. P.S. You can't load any bank of the upper 64 unless 61438 is set 0.

****

DOING IT THE HARD WAY
OR DOING IT EASILY

We still get disks formatted for 6809 by 6502 DOS commands (capitals in titles give them away), and handy-dandy instructions to drop a language, leave 6809, and issue a DOS command in 6502. We gave up that dreary nonsense six months ago. In 6809, we have RENAME, SCRATCH, and MOUNT as system commands in the mED wherever used. We also have a mED COPY command which copies/renames individual files in the same general format as the 6502 command. We then need know but five DOS commands in 6809, and to left below they are, all simple five of them. You need to remember only that the destination always is on the left and to use CAPS in the command. Why drop a program and language and go to 6502 to issue that long 6502 'copy' command because you can't remember C1=0? Saints!

```
Destination always is on left of '='

COPY a disk to another:  C1=0

DUPLICATE (backup):      D1=0

NEW (format or header)   N1:diskname,22
disk number 22:

CONCATENATE files: C1:big=0:tiny,1:teeny,0:weeny
(Appends to 'tiny' files 'teeny' and 'weeny' as
      file 'big' on drive 1)

VALIDATE (collect) a disk: V or V1
```

For a long time, those $#**! opens and closes in mBASIC, and the horrid programs you had to write in mPASCAL and in mCOBOL to be able to issue DOS commands had us dropping back into 6502, but no more, no more.

That was BS (before Swift). When Jim discovered that he could issue any DOS command from the microEDITOR (all languages but APL) by putting 'g ieee8-15' in front of the DOS command, everything turned simple, simple, simple. We guess the readers (many of them) haven't tried it--or don't believe a 'get' will 'put' a command to the DOS. Well, it will. Everywhere, in all languages but APL. Why do things the hard way? The command at left, given at the command cursor in mED, does a backup TO disk/1 of disk/0 fast as scat. You should drop into 6502 to do THAT? We can think of only one

```
g ieee8-15.D1=0
```

minor question that arises in using Swift's 'get': what do you do about spaces in filenames? Use quotes: g 'ieee8-15.N1:disk name,22'. Handling DOS commands in 6809 is now as easy as falling downstairs. Quod Erat Demonstratum.

When disks come in with all filenames in caps (inexorable if you work in 6502), pray tell what microprocessor are they for: 6809 or 6502? And in what language? Okay, maybe you LIKE to have files for eight languages/facilities (including WP) all in upper case, and maybe you're psychic and can tell where they run....

****

TRIPWIRES, MINES, HIPPOPOTAMI AND OTHER APL ACULTURATIONS

As one of our unhappy readers, who discreetly remains anonymous (and calls himself the Curmudgeon) has several times said, in frustration and in obvious rage, 'APL handles matrices with the grace of a fin de siecle ballerina, but the housekeeping in that language is as graceless as a waltzing hippopotamus. And instructions are as hard to find.' Nobody has clearly defined the the peculiarities of APL as implemented on SuperPET. So we've logged both the tripwires and our waltzes with the hippo. Herewith a few excerpts....

Day 41. A booby trap. Try to save a workspace (WS) named EXAMPLE to disk 1. Say)SAVE DISK/1. Error. Say (as in all other languages))SAVE 'DISK/1.EXAMPLE'--and weep. WS is renamed 'DISK/1.EXAMPLE' and saved to drive 0. In this mad language, you must rename the workspace with)WSID DISK/1.EXAMPLE. Then say)SAVE, and off to drive 1 goes the workspace, under the filename of EXAMPLE. Mad! A WSID is not a WSID if it's a filename going to drive 1.

The Shattered Weekend in the Minefield: We received a number of letters asking how to set up ASCII printers from APL (which we usually do from the system disk with a load module), and budgeted an hour on Saturday morning to do it. Ha! If you refer to the Character Code Tables, Appendix C, APL Manual, you'll find them in hex (do you ever send printer code in hex?). So we converted them to decimal for convenience (see below), and then ran into the INDEX problem. In all other languages, your OPTION BASE (whether 0 or 1) for matrices does not affect your system commands [char(1) in mFORTRAN, for example, remains char(1)]; APL, however, loads with ⎕IO set to 1, so any code you send to printer is off by one. If you put to printer ⎕AV 1, you send NULL or chr$(0), which, after all, is the very first code in the ASCII series, is it not? Until we sorted this out, our poor printer climbed walls, performed Immelmann turns, and wept a lot.... Is it any wonder those letters came in?

Having that in hand, it's easy to set printer with the usual ASCII codes if you use an ASCII table for any code sent above CODE 31. ESCAPE G, for example, goes to printer as 27 71, using the ASCII code for G, not the APL code for G.

The terminal screen is a different story. If you look on page 105, APL manual, you'll see a series of overstruck APL characters, starting at CODE 14 and running through CODE 31. As shown in the table below, they can be printed to screen by using ⎕AV[n], where 'n' is the code number (assuming ⎕IO is zero).

A happy note: All the conventional SuperPET ASCII codes from 0 through 13 can be sent to screen as commands using ⎕AV[n], and there they perform the exact same function they perform in all other SuperPET languages. ⎕AV[12], for example, clears screen and homes cursor, just as it does everywhere else in SuperPET. But--⎕IO must be zero, and THAT may louse up functions in WS. So, APL provides

the TC commands (p. 82, APL manual) which control printing independently of the value of ⎕IO.

TABLE 1. THE FIRST THIRTY-TWO CODES IN APL, IN DECIMAL, FOR TERMINAL SCREEN

CODE/FUNCTION OR CHARACTER GIVEN AS N IN ⎕AV[N]		CODE/FUNCTION OR CHARACTER GIVEN AS N IN ⎕AV[N]	
1	HOME CURSOR	17	GRADE UP
2	RUN	18	REVERSE/ROTATE
3	STOP	19	TRANSPOSE
4	DELETE RIGHT	20	WITH 18, RVS/ROTATE
5	INSERT SPACE	21	LOGARITHM
6	ERASE TO END LINE	22	LOCKED FUNCTION
7	CURSOR RIGHT	23	EXECUTE
8	CURSOR LEFT	24	FORMAT
9	TAB	25	SCAN/EXPAND
10	LINEFEED, CURSOR DOWN	26	REDUCE/COMPRESS
11	CURSOR UP	27	LAMP (COMMENT)
12	CLEAR SCREEN, HOME CURSOR	28	INPUT
13	CARRIAGE RETURN	29	!
14	⍱ LOGICAL NOR	30	MATRIX INV/DIVIDE
15	⍲ LOGICAL NAND	31	I-BEAM
16	⍒ GRADE DOWN	32	SPACE

◇ APL CONTROL AND CHARACTER CODES FOR ASCII PRINTER, DECIMAL ◇
CODES FROM 32 THROUGH 126 ARE IDENTICAL FOR SCREEN AND PRINTER

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	*NULL*	*LF*	*DC4*	*RS*	∨	2	;	¯	*	⊂	*D*	*N*	*X*
1	*SOH*	*VT*	*NAK*	*US*	∧	3	×	∇	?	←	*E*	*O*	*Y*
2	*STX*	*FF*	*SYN*	*SPC*	≠	4	:	Δ	ρ	⊢	*F*	*P*	*Z*
3	*ETX*	*CR*	*ETB*	"	÷	5	\	ι	⌈	→	*G*	*Q*	{
4	*EOT*	*SO*	*CAN*	)	,	6	_	∘	~	≥	*H*	*R*	⊣
5	*ENQ*	*SI*	*EM*	<	+	7	α	'	⍋	-	*I*	*S*	}
6	*ACK*	*DLE*	*SUB*	≤	.	8	⊥	⎕	∪	◇	*J*	*T*	$
7	*BEL*	*DC1*	*ESC*	=	/	9	∩	|	ω	*A*	*K*	*U*	
8	*BS*	*DC2*	*FS*	>	0	(	⌊	⊤	⊃	*B*	*L*	*V*	
9	*HT*	*DC3*	*GS*	]	1	[	∈	○	⍎	*C*	*M*	*W*	

REVERSE FIELD CHARACTERS ARE FOUND AT CODE PLUS 128
CONCAT CODE NUMBERS ON ABCISSA (TOP LINE) WITH NUMBERS ON ORDINATE (LEFT MARGIN)
TO DETERMINE ASCII CODE NUMBER OF CHARACTER/CODE AT CHARACTER LOCATION

Use the first table above to send codes to the SCREEN only. Use the second table as a reference to the ASCII codes to PRINTER through code 31, and the remainder of table 2 as a reference to the CHARACTER you will print if that code number is sent to either to PRINTER or to SCREEN.

Below find an APL function to control margin set on a DIABLO (Commodore 8300P) printer. Since the new margin cumulates on the old, you have to zero out the old margin first. (Code to do it is in ZMARGIN, below.) The rest is straightforward. Any other printer control codes may be sent in the same manner. Note that ⎕IO is set as a local variable to avoid wrecking other functions.

Call the function with: SETMARGIN n (where n is the margin). No quotes. Wilma, George, and Ted, you ruined ye ed's Saturday, but here's how to control an ASCII printer in APL in SuperPET. Stop writing those letters. And we did get some APL tables in decimal out of all this.

```
∇SETMARGIN[□]∇
[  0]     SETMARGIN MARGIN ;□IO;ZMARGIN;SPACES;SMARGIN
[  1]     □IO←0      ⍝ SET IO LOCALLY TO ZERO OR PRINTERS GO MAD!
[  2]    ZMARGIN←□AV[27 13 80]
[  3]    SPACES← 1 ↓MARGIN⍴□AV[32]
[  4]    SMARGIN←□AV[27 57 13]
[  5]    'IEEE4' □CREATE 3
[  6]    (ZMARGIN,(,SPACES),SMARGIN) □PUT 3
[  7]    □UNTIE 3
```

****

MICROPIP : A SOFTWARE REVIEW
by Dick Barnes, Terry Peterson, and Steve Zeller

All three of the authors purchased microPIP and have used it for several months. To provide a broad point of view, we circulated a draft and all three took turns revising. The general reaction: "The more we use PIP, the better we like it." Terry Peterson adds, "PIP manages files better than ever before on CBM equipment, and you can quote me as an 'old hand' at CBM disks."

MicroPIP is a Waterloo program designed to handle all files (including printer, serial, disks, and terminal) on both output and input. It operates with mainframes running HOSTCM. Steve Zeller has limited experience using it with a mainframe. It enables entry of the MONITOR, operates in dumb terminal mode, (and, in tandem with proper software, in smart 6809 terminal mode). It copies, renames, formats, searches and compares, and will log results to printer and to screen. It has the valuable ability to read and put to disk as a load module any part of SuperPET's memory with relatively simple commands. It handles files on several separate disk drives (and on a Host computer). All old DOS commands may be sent in that format rather than in the more powerful, but more complex, PIP format. With few exceptions, the manual is well illustrated with specific, exact examples (though we wish there were more on command files).

1. COPY and RENAME. These commands, combined with a very powerful implementation of the wild cards available in the Commodore DOS, allow you to copy or rename any particular filenames (or any filenames beginning, containing, or ending in specific characters), and of particular types of files (SEQ, PRG). Whilst copying, you may also rename the copied files. After we caught on (it took a bit of time), we renamed and copied every file on ISPUG disk 1 in about 10 minutes, using the powerful, general commands with wild cards. See examples below.

```
copy disk/1=disk/0
```
(copies all SEQ [default type] files on drive 0 to drive 1)

```
copy disk/1=disk/0.*,*
```
(copies all files, of whatever format, to drive 1)

```
copy disk/1.????????????????as=disk/0.*b
```
will copy to disk 1 all SEQ files on drive 0 if they end in 'b' (in this case they ended in '.b'), and renames them with the suffix of '.bas'. Thus an old file, 'proc dos.b', is renamed 'proc dos.bas'. You MUST use a total of 16 '?' and other characters in this command to pick up all filenames. The extra

copy disk/1.*,seq=disk/0.*,* (copies all files, of whatever type, to drive 1 as SEQ files) [See important note, below!]

'?' wildcards, if not needed, are discarded by the system. We copied and renamed 39 files in one 3-minute pass, but needed some two hours, the first time, to nail down the proper command. The manual says that * represents 0 or more repititions of any character, but when we did *as=disk/0.*b, instead of using ??...? in the example above, 'proc dos.b' became 'proc dosas'. You must test carefully to learn to use the wild cards.

Your files must be systematically named if PIP is to save your time. PIP can neither rename nor copy selected files en masse unless those files are identified in a distinctive way. Steve Zeller has several times suggested an ISPUG filename convention; one is in draft now. Note that if your filenames carry a distinctive prefix or suffix, you can sort, copy, and rename not only by language but by the function the file performs, as in the suffix ':bd', which, when used as an identifier with the name of the file, tells us that the language is microBASIC ('b') and that the file is a demo ('d') for the Gazette. You can locate, copy, or rename all 'filename:bd' files easily with PIP.

Initially, Dick thought that PIP converted SEQ files to PRG files, and vice-versa (see last command, left, above). But files so renamed stay in their old format, despite their new names. PRG files are still PRG files, even though renamed as SEQ files, as Terry points out. Be warned.

2. PIP makes it simple to handle files on two or more disk drives, as with devices 8 and 9, for example.

3. PIP does a good job of reading and sending to disk the SuperPET memory locations you specify, in modules which you can reload in PIP with GET, in the MONITOR with load, or from the language disk. Example: with SCROLL and PIP in memory, Dick saved $7f00-$8000 to disk as 'highmem' (which should have saved a load module containing SCROLL); shut down, loaded 'highmem' from 6809 menu, & then loaded PIP--and both SCROLL and PIP worked perfectly. In short, when something in memory (particularly a program modified in MONITOR) is up and running, you can save it as a load module--a decided blessing.

4. The MATCH command is powerful and most useful: it compares two files, line by line. If it finds difference between them, the lines from both files are printed to the screen, and optionally to printer. If you have a several versions of a complex program, this feature is golden in finding and comparing the differences.

5. You may create 'command files' which contain a series of PIP commands much used, and execute them with a one-line command. With such a file, we found MATCH remarkably fast. We went through a a pair of disks, one of them an old backup, in less than five minutes. Every difference was printed to screen and to hard copy--with the drive number, line number, and full text of the lines. These were SEQ files. PRG files are compared in hex. Fast, easy, and valuable.

6. We particularly like the feedback on commands. PIP prints to screen, and optionally sends to printer, exactly what it is doing with each file as it does it. You can STOP if you see mistakes. It is easy to make short tests.

7. DIRECTORIES. PIP's handling is lovely. [Peterson: understatement!] You may see on screen or send to printer a whole directory or any selected part.

Example: #di ieee4 = disk/0.*.bd sends to screen and to printer the filename of each file on disk ending with '.bd'. You can find the files you want quickly.

8. Steve notes that he can upload to a mainframe, sometimes, with the command 'copy serial=myfile.txt', but that he cannot download a file off the serial port, and that he had trouble setting up the serial port. The timeout option does not work in PIP, and although he set the speed, he had trouble in the passthru mode. He finally used setup in the main menu, before loading PIP.

9. That, in general, is the good news. Some few features need to be improved. The first: an uncontrolled, FAST scrolling of directories off the top of the screen. Here you sit with a powerful PIP, designed to handle files, and you cannot read the scrolling directory as it goes by; you hope to God you're at the right spot and press STOP. If you don't STOP at the right place, you get to call for a directory all over again. Luckily, we got a program from Canada called SCROLL. Loaded with PIP, it stops and starts scrolling at your command. We hope to publish the code for SCROLL soon.

10. We doubt much can be done now about inconsistent syntax; i.e., elsewhere in SPET, an apostrophe may be substituted for ". Not in PIP. If you use ' to enclose a filename with spaces in it, ('new file'), the ' is memorized as part of the filename. And quotes are not used as elsewhere in SPET (example, left).

mED and Languages:	PIP:
'disk/1.new file'	disk/1."new file"

11. COMMAND DEFINITIONS. The command definitions called to screen with ? whilst in PIP are in many cases too brief. We wish PIP included two or three 'help' screens with specific, concrete examples of the more complex commands. The syntax gets complicated, and you need the examples if you do not use PIP every day. The manual, thank goodness, contains good examples.

12. PIP RUNS ALONE. The command files for PIP are best created in the microEDITOR; (though you can CREATE them in PIP). Often, you need to see your files; we found ourselves loading them in mED frequently, so we finally filed PIP (without its help programs) on the language disk. We wish the writers had integrated PIP into the mED, as in the languages, to avoid constant reloading of of the two programs. But, for $75.00.... Available from: Watsoft Products, Inc. 158 University Avenue West, Waterloo, Ontario, Canada N2L 3E9.

mmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmm

IS THE SCREEN A RELATIVE FILE? Enter and run the little program below; watch what happens. See if you can figure out why the program prints and inputs where it does. Don't modify anything. Enter and run it as shown at left, even if doesn't make sense at the start. If you can't figure it out, blame Terry Peterson for sending it in, and see the last page of the Gazette, this issue.

A Useful Puzzle

```
 90 ! by Terry Peterson
100 print rec = 0, 'Hello'
110 print 'Now the cursor is here.'
120 input rec = 24, 'Well?', line$
```

++

BITS BYTES & BUGS

by: Gary L. Ratliff, Sr.
PO Box 829, Sanatorium
Mississippi 39112

Since 6809 memory is blown in my SuperPET, I cannot get to or work on my Development files, so I'll have to depart from my promise to show a way to optimize the LENGTH routine. Dr. Cowan of Waterloo has kindly given permission to publish a translation of it which --SuperPET permitting--will be printed next issue.

This column therefore considers where we have been and where we are going. I'll also introduce, at the suggestion of Fred Fuller of Clinton, IA, a quiz in each issue which will allow readers to test their understanding of the 6809 and 6502 instruction sets, and ask that readers by mail or phone tell me what they'd like to see discussed in this series.

Let us review what we have accomplished and also what you now should have mastered. In the earliest issues we showed how to prepare an assembly language program from the editor. The format of .asm and .cmd files has been explained. We showed how to use the editor to modify files to save time. Some of the addressing modes of the 6809 have been introduced, with examples of various ways to deal with text. And, finally, we showed how to use standard library routines.

In the last issue you should have caught the error. We showed that error as the absence of leas 2,s. We had, however, placed two parameters on the stack and each parameter required two bytes. We should have removed both paramaters by using leas 4,s. We hope everyone caught the error.

In other articles on the 6809 we have given you various useful tools: the first published example of a user-interrupt which dumps the screen to the printer, a detailed map of the 6809 side of the SuperPET, and a method of easily using the discovered routines in your own assembly language programs.

So much for where we have been. Now, where are we going? This installment will mark a transition; instead of covering basic material we'll cover the more advanced aspects of assembly language, one of which is structured programming. It is available in Assembler as well as in the high-level Waterloo languages, and in Assembler as elsewhere makes programming much simpler and easier.

If at all possible, I recommend that you read Software Engineering For Micros, a book written by Ted Lewis and published in 1979 by Hayden Book Company. If your local library has a good computer section, you should be able to locate a copy; if not, it is sold at most good computer stores. The book is most instructive, and will help you grasp structured programming. It is well written (there are some minor errors for which Ted assures me he takes full responsibility). Lewis' technique of software decomposition is invaluable when you tear down some of the system routines.

We'll also offer more tools for the assembly language programmer. First will be an extended monitor for the SuperPET which will be offered on the next ISPUG disk. [Ed. Let's call it EXMON for the moment. We have another improved monitor in the mill from Terry Peterson.] EXMON is the outgrowth of my work with MICROMON. See 'A SuperPET Monitor you can Bank On' in the March '83 issue of COMPUTE! for some of the limitations of the 6809 monitor. EXMON is intended to remedy most of these shortcomings. Second, I've rewritten a screen dump to send monitor and other output either to printer or to a disk file. Third, Paul Ruud, of Berkeley, Cal., has written a routine to let us redefine the keyboard. Last, Waterloo has sent additional data on the 6809-side map. Many tools are emerging to make assembly-language programming in SuperPET easier and less a drudgery.

Next, I may be writing about hardware. I normally don't do it. So why depart from strictly software solutions to problems? Because there is a definite need for such material. Dick tells me that at this moment three of our Associate Editors have SuperPETs with blown memory boards. After a bit of work, I was able to locate MOSTEK's data sheets and several informative articles on the 4116

memory chip, so I'll attempt to fix the memory board myself and report on results. If I succeed or fail, I'll report. If I fail I will be doing as well as the local Commodore repair technician....

If you have hardware experience you may be able to help me. The diagnostic program 'Test 6809 SPET' which is on the first ISPUG disk shows that the ram at UD37-UD44 is bad. A memory test via 'fill 9000 9fff 00' in all banks shows that banks 2,3,a,b don't produce correct results. The pattern revealed is 80 00 80 00 80 00 etc. There are 8 chips involved, and the high bit appears to be permanently set. At present the 4116 chip in position UD44 is the suspected chip to replace. Any advice or help is welcomed.

Now for the example for this issue. Many of you may have wondered how the structured programming construction: quess...endguess is actually used in a program. The following example is from the open command of EXMON for the SuperPET. The object is to allow the output to be redirected from the screen to either the printer or to a disk file.

```
open equ *
;the format of this command is 'op' to open printer, 'od' to open disk,
;'os' to open serial and 'oi' to open ieee4 (ASCII printer)
clr busy ; device never opened
jsr getchar
guess
     ldy #write   ;default case
     cmpb #'d     ;?disk
     quif ne
     ldy #append
     ldx #sent
admit
     cmpb #'p     ;?printer
     quif ne
     ldx #printer
admit
     cmpb #'s     ;?serial
     quif ne
     ldx #serial
admit
     cmpb #'i     ;?ieee
     quif ne
     ldx #ieee
admit
     ldd #invalmd
     jsr printf
     bra byby
endquess
sty mode
stx type
byby jsr waitcr
     jsr putnl
     rts
ieee equ *
     fcc 'ieee4'
     fcb 0
```

Here the object is to present the method of using guess...admit...endguess construction. As this example shows, it may be used as a case structure to select one option of many.

I have not underlined routines normally requiring it because this example is not to be run.

This is a technical error. But there is a more common error in this routine which you are to try to find.

The example is from a much larger file and the cmd file to corectly link it is not needed.

Here invalmd is the ROM area which prints: 'invalid mode'. This case is reached if the user fails to select one of the devices. As given in the outline of the instruction.

Waitcr is a monitor routine which waits until the user hits the carriage return.

;this is not in ROM as others

```
mode rmb 2
typ  rmb 2
busy rmb 1
sdsv rmb 1
     end ;
```

All the other areas are found in the area of ROM near DevList. See Map in issue seven.

+++ Now for the promised quiz. Answers next issue: +++

1) What is the difference between the action of the carry flag in the 6809 and the carry flag in the 6502 micro's?
2) List at least three methods of clearing the 6809 carry flag.
3) What is the difference between the instruction ABX and LEAX B,X?

The error in the example presented: either stx 'type' must be changed to 'typ' or the rmb 'typ' must be changed to rmb 'type'. Variables clearly don't match.

STEVE ZELLER IS MISSING.... The APL Exchange does not appear this issue because Steve's SuperPET has blown the upper 64. We'll try to make it up to you next issue.

* * *

THE PUZZLE : A POWERFUL ANSWER

```
*Fortran demo:  screenfile.fd
   character line
write(rec = 0) 'Hello'
print, 'Now the cursor is here'
write(rec = 24) 'Enter string'
read, line
print, char(12)
write(rec = 12) line
end
```

The answer to Terry Peterson's puzzler: indeed the screen (or more properly, 'terminal'), is treated as a 'relative' file, not only in mBASIC, but in microFORTRAN as well. See program at left, which places all write(rec = n) lines on the screen at the lines indicated. Note there is an offset: row 1 of the screen is line 0 and the offset persists to row 25 (line 24). This is far more than a trick; it provides powerful and simple cursor control. We suspect the same technique will work in mPASCAL and in APL. If our answer is still confusing, change the rec = entries in either mBASIC or mFORTRAN versions and watch the screen lines change. If you change to OPTION BASE 1, we'd guess line 1 no longer will be rec=0. And note you now have a powerful way to <u>test</u> relative files where you can see them.

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	No. 9: $2.75

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. Make checks to ISPUG

==========

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:_____

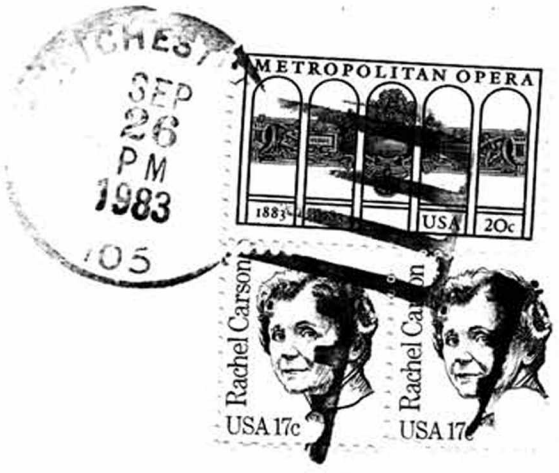
Street, PO Box City or Town State/Province/Country Postal ID#

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. Super-PET 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