# Pro/file Updates

## The Newsletter For ZX Pro/file Users

Vol. 1, No. 1 January 1984

### A HEARTY WELCOME!

Welcome to Pro/file Updates! Charter subscribers are members of a very exclusive group, indeed: 123 people world-wide. Keep the faith, numbers will grow--especially if you tell your friends about ZX PRO/FILE.

This newsletter is going to provide you with modifications and ideas to make ZX PRO/FILE serve you better. A heavy slant will lean toward "programming tips" as well. ZX PRO/FILE is versatile enough to work in many applications. If you find a particularly good use or if you customize it to meet your own needs, tell UPDATES about it. At least 123 other users would like to know.

### A MYSTERY ARISES

Some people report bugs in ZX PRO/FILE such as report-4 (out of memory) errors, report-5 (not enough room on screen) errors, and during SAVEs, some report that the computer goes ape. All problems exhibit a shocking degree of non-uniformity except for one thing: 16K Memotech RAM packs were used in every case.

I have tested ZX PRO/FILE using this memory and, mysteriously, I've not been able to duplicate any of the errors mentioned. If you use the 16K Memopack and unusual happenings occur, suspect the memory. Problems appear to go away when you reduce file capacity. 64K RAMs made by Memotech have so far worked flawlessly.

Report your difficulties using different RAM modules. Others may have already discovered a solution.

### NEW USES Dept.

Jim Cripps of Central Islip, NY has compiled a large foreign language translation data base using ZX PRO/FILE. His family uses it to improve their language skills in any of the five different languages held in the program. For each word, a two line file is created. The first line holds the initial of the language being used. Following this is the English word. On the second line of the file the foreign word is printed. A sample file would look like this:

```
*F  GOOD
BIEN
```

To use the data base as a language tutor, Cripps types "F GOOD" as a search command. This is asking the computer to give the French translation of the English word "GOOD". Before Cripps presses ENTER to find the desired file, he guesses the appropriate translation. Notice that "F GOOD" results in only the French translation. "I GOOD", "R GOOD" "S GOOD" or any other command would yield a different language. Using 64K, you could expect this data base to hold roughly 2100 words in any language(s) you desire.

PRO/FILE 2068??

Sadly, ZX PRO/FILE will not work on the new TS2068 without a substantial re-write of the machine code. Even general concepts must be changed. Watch for a 2068 version in 6-8 months. By necessity, it will be accompanied by an entirely new text as well.

GLARING OMISSIONS Dept.

Far and away, the most common problem experienced by ZX PRO/FILE owners is enlarging the D$ array to make use of a larger memory pack. It seems that the dummy who wrote the book was so concerned with getting the program lines on page 42 printed correctly that he completely forgot to tell you to RUN 4000 before going to line 17. Anyway, Joanna Grammon of New York City sent these directions that will increase capacity to 42002 bytes.

(Numbers 1-3 should be carried out immediately after powering up the computer)

1. POKE 16389,255
2. NEW
3. PRINT PEEK 16389 (answer: 255)
4. LOAD "ZX" (16K version)
5. Turn on tape recorder, then ENTER.
6. After getting Main ZX Menu, hold SHIFT and press "1". This deletes the quotes on the bottom line.
7. Press STOP, then ENTER.
8. Type CLEAR and ENTER.
9. Add the following program lines from page 42 of the PRO/FILE manual.

```
4000 DIM D$(10505,4)
4010 LET L=PEEK 16400+256*PEEK 16401
4020 POKE L+1,41
4030 POKE L+2,164
4040 POKE L+3,1
4050 POKE L+4,38
4060 POKE L+5,164
4070 POKE L+6,0
4080 LET P=0
4090 LET S=1
4100 LET C1=3
4110 LET C2=11
4120 LET E=0
4130 LET Y$=""
4140 LET Q$=D$( TO 32)
```

10. After checking program lines, RUN 4000.
11. You will now be in the program listing. To get back to the menu, GOTO 17.
12. Type "A" which puts you into ADD mode.
13. Stop the cursor on line 1 and input in reverse graphics:
    (space)SEARCH IS COMPLETE
14. Close the file to return to Main Menu. SPACE OPEN should say 42002 SLOTS.
15. If you held your mouth right, you should have done these steps with little or no difficulty. NOW, before doing anything, make 1-2 copies of the 64K expanded program. (saving time is +20 minutes--use a long enough tape!)

BYTE-BACK PRINTER INTERFACE

Al Rapp of Blowing Rock, NC writes:

I am using the RS-232 board in my Byte-Back modem and Byte-Back's printer driver program to drive my Star DP8480 serial printer. I can copy the whole screen by using USR 8347 at line 3020 of PRO/FILE but I can't select lines to be printed.

---

You must alter PRO/FILE's printer code:

```
POKE 16674,6
POKE 16675,0
POKE 16676,205
POKE 16677,93
POKE 16678,32
POKE 16679,42
POKE 16680,14
POKE 16681,64
POKE 16682,195
POKE 16683,126
POKE 16684,32
```

Also, change BASIC lines so they read:

```
3010 POKE 16675,C2
3020 RAND USR 16674
3036 GOSUB 3500
```

Finally, add these BASIC lines:

```
3500 CLS
3510 PRINT
3520 RAND USR 8351
3521 RETURN
```

This routine works when Byte-Back's printer driver is located at address 8347 for LCOPY

## THE INCREDIBLE "NOT" SEARCH

These changes allow ZX PRO/FILE to perform "NOT" searches. This is equivalent to printing all files EXCEPT the word you input as a search command. PRO/FILE's built-in multi-word search mode is used in this routine. The computer searches all files for a match to the first word. If one is found, it then scans the file to see if it also holds the second word. The file is printed only if the second word is not found in the file.

To activate the "NOT" search, place a "/" in the last position of a multi-word search command.

| | |
|---|---|
| 1983/MARCH/ | will find all files that have the word "1983" in it EXCEPT those that also contain the word "MARCH" |
| */OHIO/ | results in all files EXCEPT those with "OHIO" in them. |

Possible applications: club membership list (send newsletter to all members EXCEPT those who haven't paid their dues); abstracts (all articles on a given subject EXCEPT those by a specific author); customer follow-up (send a subscription form to all customers of ZX Pro/File EXCEPT those who have already subscribed). Use the "NOT" search to find anomolies in your test results--list all cases in category 1 EXCEPT those with the predicted result. In fact, you can combine this search with File Tabulation and Ordered displays to reveal many fact about the statistics or test data stored in the program.

If you run a 64K version of PRO/FILE, these changes should fit without reducing capacity. On 16K programs, however, you must reduce the D$ array by 100 bytes. Add or otherwise change the following lines so they read:

```
  35 IF X$="SAVE" OR LEN X$>28 T
HEN GOTO 18
  36 IF Y$<>"R" THEN LET N=X$(LE
N X$)="/"
  40 IF Y$<>"R" THEN LET X$=X$(
TO LEN X$-N)
  42 LET X$=X$
  44 PRINT X$
 121 IF PEEK 16883 AND B<>1 THEN
 IF USR 16780=N THEN GOTO 115
 125 PRINT X$+("-NOT " AND PEEK
 6883 AND N);TAB 0;"FILE SEARCH"
;;;TAB 1;"*";
1100 GOTO 42
```

One oddity of the "NOT" search is that if you use the "*" as the first word of a search command, the first file to be displayed will say, "SEARCH IS COMPLETE" even if the search has only just begun. It's a good practice to press ENTER a second time whenever this file is printed just to make sure that the search really is complete.

## PROGRAM CHANGES and DATA CAPACITY: A Symbiotic Relationship

Memory space in a computer--regardless of how much you have plugged in--is allocated for certain jobs. Two major sections of memory that concern ZX PRO/FILE are the program area of memory and the variables area. When PRO/FILE runs, all files are stored in the variables area. With 16K attached, the files (also called data) go into an array of characters: D$(11000). Since the total amount of memory is limited to 16K by the size of the rampack, most of your memory is used to store the 11000 characters of D$. The remainder is used to hold the actual program and a few other things like the system variables and display file.

When you have a fixed amount of memory--16K for example--and 11K is reserved for data storage, you only have 5K left for every thing else. Roughly, you can figure that the computer needs 1.5K just to operate. This memory is used for calculations, system variables, the TV display, etc. All that's left for the actual program is 3.5K and ZX PRO/FILE uses just about all of that. The more data capacity you have, the less program lines you can hold. A program full of files is no different than one that is almost empty in this regard. It is capacity (the size of D$) that is important.

If you want to add more than a few short program lines to PRO/FILE, memory problems will likely appear. Symptoms include report-4 error halts while the program is operating. This usually happens when the computer is executing a line that requires a little extra RAM SPACE. Inputting long 28 character search commands or moving the edit cursor are common culprits.

Sometimes the computer gets so full that it won't let you add a new program line. If you try to enter a new line but every time you press ENTER the line just sits at the bottom of the screen, lack of memory is probably the cause. This can happen when you try to EDIT an existing line too--especially if the line you're trying to edit is a long one.

You can buy a larger memory pack to expand both data capacity and program space. You can also sacrifice some of your data size to gain programming room. If your modifications are well written, the cost in terms of data capacity is usually offset by your new capability to access the data you have left with greater versatility.

Books tell you that it is a simple matter to expand an array like D$(11000). Just re-DIM it to D$(12000) to gain 1000 more characters. To make it smaller and thus free up some memory space, go the other way: DIM D$ (8000) for example.

There's just one hitch. If you have files typed into the array you lose everything when you redimension it. Since adding files is slow, tedious, and an awful nuisance, you should avoid adding them twice like you'd avoid the plague.

Fortunately there is a way to shorten or lengthen an array and still retain the data held in it. The following procedures detail exactly how to do it. Admitedly, they are somewhat involved, but if you have a program full of data, they beat the heck out of retyping everything back in again.

Use these procedures any time you need them. They can be used on any memory size with a minimum of sacrifice. Most of the time you will lose nothing when you cut back. Only when D$ is almost full will you risk losing data. You do, however, have control over just which files get the axe.

If you decide to shrink capacity by 1000 bytes and your SPACE OPEN indicator tells you that there are only 800 slots left, 200 bytes will disappear from your files. Go through your files by operating the program. Delete unimportant lines or complete files. When the SPACE OPEN tells you that there are more slots left than the amount you wish to cut back, it's safe to proceed.

## REDUCE CAPACITY WITHOUT LOSING DATA

Numbers given in small type are the results I obtained when a "stock" PRO/FILE was used with files added so that P=1242. The results you obtain will be different.

1. With PRO/FILE in the computer, exit from the Main Menu. Press SHIFT 1, STOP, and ENTER.
2. Type PRINT P and ENTER. Write down the result that comes on the screen. (1242)
3. Type PRINT LEN D$ and ENTER. Write down this result as well. (11000)
4. Now decide what you want your new length for D$ to be and type: LET P= this new length. (8000)
5. Go back into the program by typing GOTO 17 and ENTER.
6. Add a file consisting of just the asterisk followed by a character unique to the data. (I used an inverse "$")
7. After you close the file, look it up. Use the "*" and the character you chose above as a search command.
8. When the file is printed on the screen, exit from the program again--this time from the Display Option Menu--using the method shown in Step 1.
9. Type PRINT PEEK 16507+256*PEEK 16508 The result you get is the address in memory where this file starts. Jot down your answer. (29347)
10. Type PRINT PEEK 16400+256*PEEK 16401 This result is the address held in VARS, the system variable which holds the address of the beginning of the variables area of memory. Since D$ is the first variable entered, the address shown is the beginning of D$--not the actual data, but the pointers which define the variable's type, name, and length. Record this address. (21341)
11. Take your answer in Step 10 and peek it's value. Ex: PRINT PEEK 21341 The value you get should be 201 which is translated by Sinclair Basic to signify the name of the first variable in memory. 201 means D$. (this is another "blind faith" principle)

12. Now take the new length decided upon for D$--8000 using my example in Step 4--and type RAND (this length). Ex: RAND 8000. This action converts the new length for D$ into a 2-byte integer. The result is stored in SEED, the system variable that is set by the command RAND. SEED is located at addresses 16434 and 16435. (credit for this trick goes to the Timex User Group of the Boston Computer Society, June '83 newsletter)

At this point here is what we know:
* The first byte of D$ (21341)
* The length that D$ is to become (8000)
* The address in memory where this new D$ will end (29347)

The first five bytes after address 21341 are pointers which define the characteristics of D$. You can poke these bytes to change its length. The values you give them come from SEED which you just set by typing RAND. The first pair of bytes must hold the new length of D$ plus 3. The next byte holds the number of dimensions, and the last pair of bytes holds the actual length of D$. So now you POKE:

If the number held in VARS is 21341, you would enter:
POKE 21342, PEEK 16434+3
POKE 21343, PEEK 16435
POKE 21344, 1
POKE 21345, PEEK 16434
POKE 21346, PEEK 16435

Now the computer thinks that D$ is 8000 characters long, but there is still the matter of cleaning up the old bytes of D$. This is done by creating a new array, the length being equal to the left over bytes. This would be the original length of D$ minus the new length. Using my example, the length of the new array should be 11000-8000 or 3000. Thus,

14. Type RAND (the old lenght minus the new length) Ex; RAND 11000-8000
This action puts the length into SEED so that you can create a new array by poking.

The answer you arrived at in Step 9 will be the first byte of this new array. This address holds the name. If the name for D$ is 201, then 202 must be the name for E$. (more blind faith) Therefore using my numbers as examples, you would POKE:

15. POKE the address you arrived at in Step 9 with 202. Ex: POKE 29347,202
16. Now the five bytes that follow (29348 to 29352) are poked just as in Step 13.
POKE 29348, PEEK 16434+3
POKE 29349, PEEK 16435
POKE 29350, 1
POKE 29351, PEEK 16434
POKE 29352, PEEK 16435
17. Cross your fingers because now you are going to find out if you did everything correctly. You should have a shorter D$ and a new E$. Find out by typing:
PRINT LEN D$ (8000)
PRINT LEN E$ (3000)
18. To free up the memory that E$ occupies, make it smaller by typing: DIM E$(1)
19. Reset the variable P to its original value. Type: LET P=your answer in Step 2.
20 Go back into the program and check your files. GOTO 17 and ENTER.

Everything is intact. The SPACE OPEN indicator reveals that there is less file space. You have more room to add extra program lines. It is important to remember that you have an E$ array in memory now. You can utilize it in your modifications. If you ever reduce capacity again, be sure the new array you create is named something different. In Step 15, use 203 to create an F$ instead of another E$. You might crash the system if you have two arrays both with the same name. I'm not sure.

## UPGRADE TO A LARGER MEMORY WITHOUT LOSING ACCUMULATED DATA

If all this "pointer poking" seems more like a combination adventure game in the memory caverns of your computer and Russian Roulette, hang onto your hats. Here's how to use the same idea to lengthen an array without losing data. These steps convert your present 16K program into a full blown 64K data base.

1. Load the 16K PRO/FILE into the computer that has had Ramtop poked to reflect a full 64K of RAM.
2. Break into the listing following Step 2 of the previous section.
3. Type PRINT P and record your result. (1311)

4. Type DIM E$(1000,31) and ENTER.
   This action reserves 31000 bytes for data.
5. Enter RAND (PEEK 16404+256*PEEK 16405)-(PEEK 16400+256*PEEK 16401)-4
   This sets SEED to the total number of bytes in the variables section of memory. This entire space is about to become part of the D$ array.
6. Now type PRINT PEEK 16400+256*PEEK 16401. As in Step 10 of Data Reduction, this answer is the first byte of D$. The 5 pointers following this address can be modified so as to consume the entire variables area of RAM. Record your answer.
7. Poke the 5 consecutive bytes beginning with your answer in Step 6 plus one.
   POKE 21342, PEEK 16434
   POKE 21343, PEEK 16435
   POKE 21344, 1
   POKE 21345, PEEK 16436-3
   POKE 21346, PEEK 16435
8. Cross your fingers again and type:
   PRINT LEN D$
9. In the Data Reduction routine, you had to clean up extraneous bytes left by shortening D$. In Data Expansion, you must re-enter all of your other variables. When the array was enlarged, it literally consumed the other variables: P, Q$, E, X$, etc. All of the pointers, characters, and numbers are still in memory, but they're not variables any more; they're now part of D$. If you're curious, type PRINT D$(10950 TO) to see what variables really look like.

   Re-enter the variable p. Type LET P=its old value given in step 3.
10. Then enter the remaining variables:
    LET Q$="  32 spaces  "
    LET C1=3
    LET C2=11
    LET S=1
    LET E=0
    LET Y$=""
11. Type GOTO 17. You are now the proud owner of an enlarged data array and you didn't have to retype a single file. Your old data is still there!

WHAT IS GOING ON HERE?

If all these RAND's and POKE's leave you cold, here is a different way to look at what's going on in the data modification process. Think of an extension ladder. Each rung of the ladder is a byte of memory.

The D$ array is often considered to be a large contiguous block of memory much like you'd think of the whole ladder. But, the ladder could also be thought of as having 2 parts: the lower half and the upper extendable portion.

In data reduction, you take the 11000 character array (an 11000 foot ladder) and say, "This is really 2 separate arrays--one 8000 characters and the other 3000 characters in length.

The RAND's and POKE's are what do this. It's like scratching out the label on the ladder that reveals its length. This may seem like six of one and half a dozen of another, but now comes the "coupe d' grace". We can release the extension ladder and lower it down to make it one shorter ladder. This is carried over to Sinclair Basic when E$(3000) is redimmed to E$(1). In one fell swoop 3000 bytes are gone from the array. The memory space is available for other uses. Since it was E$ and not D$ that was redimmed, all the data in D$ remains untouched.

A ladder analogy can be used for data expansion too. In this case you must also take into account the program's other variables (P, C1, S, Y$, Q$, etc.). Imagine a ladder that's a little too short propped up against a building. This ladder is the initial D$(11000). Hanging from the top rung is a paint bucket which is full of the other variables used by the program.

Our worker has painted as high as he can reach. To go still higher he obtains a second ladder and secures it to the top of the first. This is what you do when you follow Step 4 of the expansion routine.

The paint bucket full of the program's other variables which used to be at the top of the ladder is still there but it now is only midway up. The bucket is of little use until it is carried to the very top (done by Steps 9-10 in the expansion procedure).

## STRINGY FLOPPY NOTES

If you use a CAI Stringy Floppy for high speed program saves, you're not alone if you tried unsuccessfully to adapt PRO/FILE to work with this device.

With the help of Don Bernath in Michigan Peter Danes in New York, Jim Benedict of Miss., Lionel Barthelemy of Alabama, and Frank Finkelstein of New York, this 16K Stringy modification emerged.

Add or alter the following lines of a CLEAR copy of ZX PRO/FILE so they read:

```
  17 GOTO 9000
  25 IF X$="SAVE" OR X$="LOAD" T
HEN GOTO 9100
  35 IF X$(LEN X$)="/" OR LEN X$
>28 THEN GOTO 18

4000 DIM D$(10500)
4080 LET P=20
4090 LET S=1
4100 LET C1=3
4110 LET C2=11
4120 LET E=0
4130 LET Y$=""
4140 LET Q$=D$( TO 32)
4150 LET D$( TO 21)="*SEARCH IS
COMPLETE*"
160 GOTO 18

9000 CLS
9010 IF PEEK (PEEK 16400+256*PEE
K 16401)<>128 THEN GOTO 18
9040 POKE 16390,2
9050 CLEAR
9060 POKE 16451,8
9070 DIM D$(10700)
9080 RAND USR 10243
9085 FAST
9090 GOTO 9000
9100 CLS
9105 IF X$="LOAD" THEN GOTO 9020
9140 POKE 16390,2
9150 POKE 16451,6
9160 RAND USR 10243
9170 GOTO 9085
```

Once the changes are made save a copy on a wafer using the ESF menu. The program should be labeled File 1. To initialize all the variables type RUN 4000. When you are ready to save data, type SAVE from the Main ZX Menu.

Once a program and data file have been put onto a stringy wafer, Load the program by using the ESF Menu to load File 1. Then, go to Basic and type RUN. File 2, the data, will automatically load.

From the Main ZX Menu you can type SAVE to place updated files on the wafer. You can also type LOAD to load other data files stored on different wafers.

Upcoming issues will include 64K Stringy modifications as well as an idea for relocating the CAI ROM software into RAM so you can use the stringy with other hardware or software that uses the 8-12K address space.

I have learned that CAI is developing a "new and improved" Stringy Floppy. To find out more about this device contact:

CAI Instruments
152 E. Saginaw Rd.
Sanford, MI 48657
(517) 687-7343

## UNCLASSIFIED

Sell that piece of gagetry that failed the smoke test, or that extra printer, or memory pack. Non-commercial ads: $5.00 for 5 lines.

FOR SALE: Gorilla Printer (1 only), new still in carton. $175 Requires centronics parallel interface. Tom Woods, P.O. Box 64 Jefferson, NH 03583

Pro/file Updates is published 4 times a year, in January, April, July, and October.

Subscriptions rate: $9.95 annually

Edited and Published by:

Thomas B. Woods
P.O. Box 64, Jefferson, NH 03583
(603) 586-7734