***** BETA DOS FOR THE PLUS D *****

- CONTENTS -

INTRODUCTION 1

CAT 2

FORMAT 3

SAVE file TO file 3

SAVE OVER 3

LOAD @ and SAVE @ 4

SERIAL FILES 5
- **Creating a Serial File** 5
- **Reading a Serial File** 6
- **CLEAR # and Clearing Disc Errors** 7

MOVE 7

RANDOM ACCESS FILES 8

- **Creating a Simple Random-Access file** 8
- **Reading a Simple Random-Access File** 9
- **POINT** 9
- **Altering a Random-Access File** 10
- **Extending a Random-Access File** 11
- **Random-Access Functions:**
  - **File Pointer Function** 11
  - **File Length Function** 11
  - **End-of-File Function** 12
- **Data Packing in Random-Access Files** 12
- **Variable-Length Records** 14
  - **POINT with OVER option** 14
- **Opening Multiple Files** 15
- **MOVE and Random-Access Files** 15

SOME DIFFERENCES FROM G+DOS 16

PEEKING THE PLUS D's MEMORY 17

SYSTEM VARIABLES 17

BACKUP UTILITY 18

COMPRESSION UTILITY 18

Amendments (added 1/7/2004 by spt)
- **Undocumented Commands** 19
  - **CAT 1""""** 19
- **Useful Poke @`s** 19

Beta Dos Bug Fixer
FIXER Listing 20

Scanned, Typed, OCR-ed, and PDF by

Steve Parry-Thomas 27 June 2004.

This PDF was created to preserve the manual for the future.

For all ZX Spectrum, Plus D
And WoS users

(PDF For Michael & Joshua)

****

INTRODUCIION

Beta DOS provides a number of major enhancements to the G+DOS supplied with the PLUS D system, without occupying any of the Spectrum's RAM or requiring you to replace the PLUS D's ROM. You can still use your G+DOS discs, and Beta DOS should be compatible with any software that does not make use of the PLUS D's internal RAM. The installation program BDM (Beta DOS Maker) merges the Beta DOS code with your existing system file, which can then be SAVEd as a complete disc operating system.

By kind permission of FORMAT, the INDUG magazine, published enhancements to convert G+DOS 2 to 2a are included in the BDM installation program; conversion is done automatically before Beta DOS's code is added to your system file. The ability to escape from keyboard lock-ups by pressing the disc interface button followed by the zero key has also been included, using a method similar to that described in FORMAT magazine.

This program requires a PLUS D with ROM version la (the ROM version is printed when you load the system file). It is normally supplied on a 3.5" B00K formatted disc, but can be supplied on tape on request.

MAKING A BETA DOS SYSTEM FILE

1. Load G+DOS as you would normally. Have ready either a newly - Formatted disc, or one without a system file.

2. Load BDM from the Beta DOS disc. It will auto-run.

3. The program will prompt you to insert a disc containing the normal G+DOS. The G+DOS system file will be loaded, converted to G+DOS 2a if required, and the Beta DOS code will be integrated with it to form a complete Beta DOS.

4. When prompted, press any key to save Beta DOS onto the disc you have ready, using the name "+sysBeta". The Spectrum will then NEW itself. RUN to load Beta DOS. In future you can load Beta DOS as you would G+DOS.

5. You might like to FORMAT some new discs using Beta DOS. FORMAT gives you the option of creating a larger catalogue, and the new disc format allows faster data transfer.

6. Beta DOS can be copied onto further discs using e.g:

   SAVE dl"+sysBeta" TO dl"+sysBeta"

   The program CANNOT be SAVEd directly from the PLUS D’s RAM using e.g. SAVE "+sysBeta" CODE, unlike G+DOS.

****

This program tool; me a lot of work to write, and the price is reasonable - please buy it, don't steal it!

If you have any questions, suggestions or problems write to;

ANDY WRIGHT
BETASOFT, 21 WHYCHE AVENUE, KINGS HEATH, BIRMINGHAM, B14 6LQ

CAT

The **CAT** command can now be used on its own to give a catalogue of files on the current, drive. Simple CATs, Such as **CAT** or **CAT 1!** or CAT 2! will work much faster than before, and automatically sort file names into alphabetical order. To be precise, CAT sorts into order according to the ASCII code, so e.g. "+" precedes numerals, which precede capitals,' which precede lower-case letters. Because simple catalogues are normally printed in 3 columns, you will get something like this on the screen:

```
* MGT PLUS D DISC 1 CATALOGUE *

+sysBeta   Snap B     prog1
prog2      prog3      squash
test

Number of Free K-Bytes == 620
7 File(s), 73 Free Slots
```

If you do not want a sorted catalogue, you can turn off the alphabetic sorting without sacrificing the higher speed, using a POKE explained in the "System Variables" section.

It is possible to alter the number of columns used by the catalogue by POKEing a system variable that holds "columns to use". If you **POKE @126,1** you will get a 1-column catalogue like this:

```
* MGT PLUS D DISC 1 CATALOGUE *

+sysBeta
Snap B
Prog1
prog2
prog3
squash
test

Number of Free K-Bytes = 620
7 File(s), 73 Free Slots
```

You might want to use e.g. **POKE @126,6** when doing **CAT #3;1!** to a printer able to print 60 or more characters per line. **POKE @126,3** to restore the normal 3-column output.

At the end of all CATs, you will see an indication of the number of files in the catalogue, and the number of catalogue "slots" still available for extra file names.

You can send a sorted catalogue to a disc file using something like this:

```
OPEN #6;dl"catalogl": CAT 6;1!: CLOSE #*6
```

Omit the "**!**" to send the more detailed type of catalogue.

Note: The improved **CAT** requires enough free memory to store all the possible file names temporarily, if this is not available, the original G+DOS routine will be used instead, giving a slower, unsorted catalogue and a "file count" of zero. A disc with a 4-track catalogue holding up to 80 files needs 800 bytes of free memory for a faster **CAT.**

FORMAT

The FORMAT command allows you to specify how many tracks are to be used for the disc catalogue, and thus how many files the disc will be able to hold. If you do not specify a value, the normal 4-trach catalogue is assumed. Each catalogue track holds 20 file names. Some examples:

FORMAT d1
FORMAT d1,4 Both allow the normal 80 files and give 7B0K of disc space free for files on an 800K disc.

FORMAT d1,5 Allows 100 files and gives 775K free.
FORMAT d1,10 Allows 200 files and gives 750K free.
FORMAT d1,39 Allows 780 files and gives 605K free.

Thirty-nine tracks is the maximum you can allocate to the catalogue, and four tracks is the minimum.

Notes: Discs PORMATed with more than 4 catalogue tracks should not be used with B+DOS. You should also not make **SNAPSHOTS** onto discs with more than 4 catalogue tracks.

As well as allowing you to specify a larger catalogue, **FORMAT** is more user-friendly. Instead of simply overwriting 6000-odd bytes at 49152 as it used to, **FORMAT** uses the free memory available to BASIC, so nothing that is in use will be overwritten. (If you get an **"Out of memory"** error, you can use **CLEAR** with a number like 65000, to raise **RAMTOP**, or just **CLEAR** to delete variables, or **NEW** - all of which give more free space.)

An improved disc format allows data to be transferred to and from the disc 10% faster than before. (Discs formatted by Beta DOS can still be used by G+DOS, provided the catalogue is normal-sized.)

SAVE file TO file

Beta DOS can copy any file type, including 48K and 128K Snapshots, **OPENTYPE** files and execute files. (If you have a single-drive system, this may require multiple disc swaps.) G+DOS overwrites important memory areas while copying, and has to do a **NEW** even if it succeeds in copying a file, whereas Beta DOS uses only free memory. This means that you might decide to CLEAR a high value (e.g. 65000) and/or NEW if you are copying files and wish to minimize disc swapping. The **SAVE file TO file** command now also tells you the file name it is loading or saving as copying proceeds - particularly useful when copying multiple files using wildcards. For convenience, you can press **ENTER** as well as **SPACE** when disc swapping, in response to **"Insert TARGET disc - press SPACE"** or **"Insert SOURCE disc - press SPACE"**.

When using two disc drives, **SAVE d1"file name" TO d2"file name"** usually stops with drive two running continuously. I have not been able to correct this problem: as with G+DOS, you need to use drive two in order to stop it. **CAT 2** is an easy method.

SAVE OVER

You can use. e.g. **SAVE OVER d1"file name"** to overwrite an existing file without the **"Overwrite?"** prompt being given. This is particularly useful when re-saving d multi-part program, or a SCREEN$ file you do not want to corrupt.

LOAD @ and SAVE @

These commands now allow multiple sectors to be handled in one go, by including a final parameter to specify the number of sectors. When sector 10 is reached, sector 1 of the next track is used, allowing multi-tract: **SAVE/LOAD**. This is much faster than using many single—sector operations. For example:

SAVE @1,10,1,16384,96

This SAVEs memory to drive 1, starting at Track 10, Sector 1 and taking the data from 16384 onwards. 96 sectors are SAVEd (48K) so all of tracks 10-18 and sectors 1-6 of track 19 will be used. This takes about 3 seconds.

LOAD @1,0,1,35000,40

This LOADs data from drive 1, starting at Track 0, Sector 1 and placing it in memory at 35000 onwards. Forty sectors are LOADed - this 20K of data comprises the normal catalogue tracks. LOADing will take under 2 seconds.

When a **LOAD @** or **SAVE @** reaches the highest-numbered track on one side of a disc (track 79 on an 80—track disc) and has to step to the next track, it will move to the first track on side two of the disc. The tracks are numbered 0—79 on side 1 and 128-207 on side 2, so the move is from track 79 to track 128. It is worth remembering this gap in track numbering when writing utilities that use these commands.

If you omit the last parameter (sectors) it Is assumed to be 1, and the commands act as they do under G+DOS, with one exceptions G+DOS has a problem with **SAVE @** - if a new track is stepped to just before the sector is written, and the disc was stationary beforehand, the sector will often be incorrectly written. This makes it, and often the next sector, unreadable until the disc is re-Formatted. Beta DOS corrects this problem.

The high speed of Beta DOS's multi-sector **LOAD @** and **SAVE @** is exploited in the **BACKUP** program on your Beta DOS disc. The program is written entirely in Basic and I encourage you to have a look at it.

SERIAL FILES

Serial files are described rather briefly in the PLUS D manual, under the heading **"Using Streams and Channels"**, but many users will be unfamiliar with some of the concepts involved. Since the random-access facilities provided by Beta DOS are closely integrated with the serial file facilities of G+DOS, you need to understand serial files first - hence the existence of this section!

Creating a Serial File

To create a serial file, you must first of all OPEN the file for output, specifing a stream number and a file name, for example:

```
10 OPEN #4;dl"testfile"OUT
```

(Beta DOS allows you to use **RND** in place of **OUT** for greater reliability - see the section **"Some Differences From G+DOS"**.) The disc operating system will check to see if **"testfile"** already exists on drive 1, and ask you whether you want to overwrite the old copy if It finds one. Stream 4 will be assigned to the file, unless stream 4 is already in use. (You can use any stream number from 4 to 15.) Prom now on, PRINT commands qualified by "#4" will PRINT to the disc file, rather than the screen. For example:

```
20 FOR n=l TO 50
30 PRINT n;" abcdefghi"
40 PRINT #4;n;" abcdefghi"
5B NEXT n
```

You can see what is being sent to the disc file because line 30 prints a copy to the screen. The file will contain 50 strings, from **"1 abcdefghi"** to **"50 abcdefghi"**. In many applications these strings may be referred to as **RECORDS.**

If every single character had to be placed immediately onto the disc, writing to a file would be very slow. Instead, characters are accumulated in a special buffer in memory until there are enough of them to fill a whole disc sector. Then the disc drive is started up, if need be, and the whole sector is written in one go. Similar buffers are used when files are read. The **PLUS D** stores 510 data characters in each sector. Two more bytes are used by the **DOS**, giving a total of 512 bytes per sector. The use of a buffer means that the program above is incomplete; when the **FOR-NEXT** loop finishes, the buffer will be only part-full, and if nothing is done the information in it will be lost forever. Therefore we need to **CLOSE** the files

```
60 CLOSE #*4
70 STOP
```

This writes the last buffer to the disc, and also creates an entry in the disc catalogue. The type is shown as **"OPENTYPE"**. The **"*"** is very important! If it is omitted the line will be accepted, but the computer will crash when the line is executed because of a bug in the Spectrum ROM, and you will lose your program.

READING A SERIAL FILE

Having created a serial file as described above, we will now read it. Again the file must be **OPENed,** but this time with the **IN** keyword. The stream number you specify in the **OPEN** statement can later be used to read data from the file using either **INPUT #** or **INKEY$ #.**

```
100 OPEN #4;dl"testfile"IN
110 INPUT #4;a$
120 PRINT a$
130 GO TO 110
```

The example above will show the file contents and then stop with an **"END o-f file"** report. To close the file, you should type:

```
CLOSE #*4
```

Although closing an input file is not as vital as closing an output file, input files use memory for a buffer just as output files do, and this cannot be reused without a **CLOSE.** Besides, the stream needs to be closed if it is to be used again.

Each **INPUT** reads one of the strings originally **PRINTed** to the file, and you may wonder how this is done - in other words,- how does the DOS know where a string ends? The answer is that each string is **"terminated"** by a special character, **CHR$ 13**, which is called **"carriage return"**, a name dating from the days of teletypes. When you enter something like:

```
PRINT "one"; PRINT "two"
```

the Spectrum actually sends **"o"**, **"n"**, **"e"**, **CHR$ 13**, **"t"**, **"w"**, **"o"**, **CHR$ 13** to a ROM routine that puts the normal letters on the screen but **RESPONDS** TO the **CHR$ 13**s by printing on the next line. Printing to a disc file is similar, but the **CHR$ 13**s are actually stored on the disc, instead of being responded to. When you come to **INPUT** from the file, the DOS reads characters from whatever point it has got to in the file until it finds a **CHR$ 13.** It then assigns these characters to the variable specified in the **INPUT** command. (The **CHR$ 13** itself is thrown away.)

INKEY$** can be used to read a disc file one character at a time. Unlike **INKEY$ from the keyboard, **INKEY$** from disc always gets a character. Try changing line 110 above to:

```
110 LET d$=INKEY$#4
```

I suggest you also add a new line:

```
90 CLOSE #*4
```

This prevents **"Stream used"** errors and is often convenient. Now **RUN 90** to try the program with the existing line 120, then with these variants;

```
120 PRINT a$;
```
or
```
120 PRINT a$;" ";CODE a$
```

The last version will show up the **CHR$ 13**s explicitly. You can create other files that contain "control codes" (which cause actions) other than **CHR$ 13.** For example, the print comma which tabulates screen output sends **CHR$ 6**s to a disc file, and **INK, PAPER, TAB** etc. send special character sequences.

INPUT LINE

If some of the strings being INPUTed contain quotes - for example, **"Bide-a-Wee"** as a quoted house name in an address (ugh!) you will need to use **INPUT LINE**, just as you would if the **INPUT** was from the keyboard. For example:

```
INPUT #4; LINE a$
```

More About Opening and Closing Files

It is possible to **OPEN** a file without specifying **IN** or **OUT**; in that case, the DOS assumes you mean **OUT** if the file does not already exist in the catalogue, and assumes you mean **IN** if it does.

As well as its use for closing a particular stream, **CLOSE** lets you close **ALL** open files using the form:

```
CLOSE #*
```

CLEAR # clears all open files, but without doing a CLOSE. This makes no difference to **IN** files, but **OUT** files will be lost.

CLEARING DISC ERRORS

CLEAR # also has an additional role on the PLUS Ds it resets a **"number of files open"** counter to zero. This counter is important, because if it is not zero and you change the disc in the drive, the free space on the new disc cannot be used properly by **SAVE**. Normally **OPEN**, **CLOSE** and the other DOS commands maintain a correct counter value, but some errors can result in odd values which require **CLEAR #** to be used. For example, trying to **FORMAT** a write-protected disc sets the counter to 255! The counter value can be read using one of the new **Beta DOS** functions (see **SYSTEM VARIABLES**).

MOVE

The **MOVE** command reads a file, a character at a time, and writes it to another file or a stream. It actually uses something very like successive **INKEY$ #**s and **PRINT #**s to do this, but the process is invisible to the user, and the streams and buffers associated with the disc files are **OPEN**ed and **CLOSE**d automatically by the DOS. For example:

```
MOVE dl"testfile" TO dl"copyfile"
```

"Testfile" must exist and **"copyfile"** should not, unless you want to overwrite it. This is a fairly slow method of copying a file and **SAVE file TO file** (which works with Serial files if you use Beta DOS) is much faster for long files. However, **MOVE** is more flexible. For example, if you had two files called **"first"** and **"second"** you could create a single longer file that contained copies of them both like this:

```
10 OPEN #5;dl"conibined"OUT
20 MOVE dl"first" TO #5
30 MOVE dl"second" TO #5
40 CLOSE #*5
```

You can also **MOVE** files, to stream 2, which is the main part of The screen, stream 3,which is the printer, and random-access files (see later).

RANDOM ACCESS FILES

Although serial files can be very useful they have major this disadvantage that to locate a particular item in a file you need to read all the previous items,. To alter an item you have to read and rewrite the entire file. For some applications, this can be very inconvenient, and the problem gets worse as the file gets bigger. Beta DOS provides **RANDOM ACCESS** filing. This means you can examine or change any part of a file without having to load the whole thing. The normal **OPENTYPE** files are used, but they are **OPENed** with the **RND** extension; e.g.:

```
OPEN #5;d1"testx" RND
OPEN #strm;d2"file name" RND
```

This allows you to write to the file, using **PRINT #**, as you could if you had used **OUT**, and read it using **INPUT #** or **INKEY$ #**, as you could if you had used **IN**. In fact you can change all the **INs** and **OUTs** in the examples for serial files to **RNDs** and the programs will work as before. (But see the section **"Some Differences from G+DOS"** for information on more subtle distinctions.)

Creating a Simple Random-Access File

Although you can OPEN any existing OPENTYPE file using RND, explanations will be more straightforward if we create a simple test file, as shown below. Those who hate typing will be pleased to learn that the examples are included on their Beta DOS disc. **LOAD dl"prog1"** now.

```
10 CLOSE #*4
20 OPEN #4;dl"test"RND
30 LET a$="abcdefghi"
40 FOR n=1 to 200
50 LET a$( TO 3)=STR$ n
6B PRINT AT 10,10;a$
70 PRINT #4;a$
B0 NEXT n
90 CLOSE #*4
100 STOP
```

Just **RUN** to create the file on one of the blank discs you formatted earlier. I have used the first 3 characters of each string to store a record number so that we can easily check which one we are looking at in our experiments. The file will consist of 200 strings or records, from **"1 defghi"** to **"200defghi"**. Although the text of each string is 9 characters long, each one will take up 10 characters of disc space because they are terminated by carriage return characters (**CHR$ 13s**). These strings are an example of FIXED-LENGTH records, and the advantages of using this system will become apparent later.

In our example, it was easy to count the characters in a$ and know how long the records would be, but when you use longer strings a better way is to use **DIM**. For example:

```
25 DIM a$(99)
```

would ensure each string was 99 characters long and took 100 bytes of disc space. All this should be plain sailing if you followed the discussion of serial files; if line 20 had ended in **OUT** everything would have worked exactly the same way. The disc file you have created is a normal **OPENTYPE** file.

Reading a Simple Random Access Files

Assuming you have an existing **OPENTYPE** file, you can OPEN it With the **RND** qualifier and handle the file just as if you had Used **IN**, reading one record after another. As you do this, an Internal **FILE POINTER** advances through the file as **INPUT #** or **INKEY$** are used. This points to the next character to be read. For example, if you **INPUT** a 10-character string, the pointer Advances by 11, because of the **CHR$ 13** terminator. (**OUT** files Have a similar pointer which points to the next position to Write to, which is always at the end of the file.)

The file pointer used by Beta DOS starts with a value of zero Before the first character is read, so we can say that the first character is at position zero in the file. Just before we read the last character in the file, the pointer mil have a value 1 less than the file length. The file can be thought of as a sequence of characters numbered from zero to (file length-1). When the last character has been read, the pointer value will equal the file length, Any further reads will give an end-of-file error. The file pointer of a serial **IN** or **OUT** file cannot be changed by the user; it simply increases automatically during **PRINT**, **INPUT** and **INKEY$**. In contrast, Random-Access files (**OPENTYPE** files **OPENed** using **RND**) allow the file pointer to be freely altered using the **POINT** command.

POINT

The syntax is;

POINT #stream,position

The file Pointer for the file associated with the specified Stream is immediately moved to the specified position. (This will often cause a new sector to be loaded from disc.)

Note: If you are using 48K mode and have forgotten how to type **POINT**, it is Extended/sym-shift 8. The appearance of listings can be improved by typing a space first, to get the indentation right.

Now try this, by using **RUN 200**

```
200 CLOSE #*4
210 OPEN #4;d1;"test" RND
220 INPUT "Record? ";rec; POINT #4,(rec-1)*10
230 INPUT #4;a$
240 PRINT a$
250 60 TO 220
```

You should be able to see the advantages of having fixed-length records - the pointer value for any record can be easily obtained from the record number, allowing you to **INPUT** from any record in the file very quickly. If the record you want is in the current sector, it will be obtained particularly quickly, but even if it is at the other end of a 700K file, and the disc is stationary, the record can be obtained in 1 or 2 seconds.

Notes: **POINT** with a value less than zero will give an **"integer out of range"** error, and **POINT** with a value greater than the file length will give an **"END of file"** error.

Altering a Random-Access File

An **OPENTYPE** file **OPENed** in random-access mode can be written to as flexibly as it can be read. The file pointer indicates the position that data will be written to, as well as read from, so **POINT** allows any record to be altered. The program below demonstrates random reading and writing of our trusty **"test"** file. It leaves the record number intact to provide reassurance, although we always know which record we are dealing with simply from the file pointer value we set using **POINT**: record=file pointer/record length+1. (Later you will see how to read the value of the file pointer directly.) Now **LOAD dl"prog2"**.

```
10 CLOSE #*4
20 OPEN #4;dl"test"RND
30 DIM a$(9)
40 PRINT "Read, Write or Exit? (R/W/E) "
50 LET c$=INKEY$
60 IF c$="W" OR c$="w" THEN GO SUB 100
70 IF c$="R" OR c$="r" THEN GO SUB 300
80 IF c$<>"E" AND c$<>"e" THEN GO TO 50
90 CLOSE #*4: STOP

100 INPUT "Record to write? ";r
110 IF r=0 THEN RETURN
120 POINT f»4,(r-1)»10
130 INPUT #4;a$
140 PRINT "Old text:";a$
150 PRINT "New text?"
160 INPUT n$
170 IF n$="" THEN BO TO 100
180 LET a$(4 TO )=n$
170 POINT #4,(r-1)*10
200 PRINT #4;a$
210 GO TO 100

300 INPUT "Record to read? ";r
310 IF r=0 THEN RETURN
320 POINT #4,(r-1)*10
330 INPUT #4;a$
340 PRINT a$
350 GO TO 300
```

I suggest you play with the program, reading and writing records all over the file. (You might want to add a check to prevent the use of record numbers greater than the number of records in the file.) Enter a **"record number"** of **0** to stop writing or reading. (In fact the write subroutine at line 100 can serve for reading as well, since if you press **ENTER** when prompted for "New text?" the record will not be altered.) Notice that **POINT** is used at line 120 to set the file pointer for an **INPUT**, and then again at line 190 for a **PRINT** to the same record, because the **INPUT** will have moved the painter. When you are finished press **"E"** to exit and **CLOSE** the file. The disc may or may not run, depending on whether you have altered the current sector or not.

RANDOM-ACCESS FUNCTIONS

FN P - Reading the File Pointer

As well as being able to move the file pointer using **POINT,** Beta DOS is able to read the current pointer position using a new function, **FN P.** (**LOAD dl"rafprog"** now.) To use the function you need to have a line like this somewhere in your programs

```
1 DEF FN p(x)=USR 8
```

(You can **MERGE dl"line1"** to obtain a line with all the new function definitions in it.) **FN p(stream)** will tell you the pointer position for a random-access file associated with the specified stream; e.g.

```
10 OPEN #4;dl"test"RND
20 PRINT FN p(4);" ";
30 INPUT #4;a$: PRINT a$
40 GO TO 20
```

This function is most useful when a file contains variable-length strings and the file pointer moves by different amounts with each **INPUT.** You can find out the position of a particular string and get back to it later using **POINT.**

FN L - File LENGTH
Extending a Random-Access File

You can extend an **OPENTYPE** file easily by setting the file pointer to the end of the file using **POINT** before you use **PRINT #** to add new data. To do this you need to know the length of the file, and Beta DOS provides a new function to tell you - **FN L.** To use it you must include somewhere in the program a line like line 100 below:

```
100 DEF FN L(x)=USR 8
110 CLOSE #*4
120 OPEN #4;dl"test"RND
130 LET length=FN L(4)
140 POINT #4;length
150 FOR n=1 TO 10
160 PRINT #4;"extended "
170 PRINT FN L(4)
175 NEXT n
180 CLOSE #*4: STOP
```

This will add some data to the end of our much-used **"test"** file, printing the current file length as each string is added. If line 140 had been omitted, the first part of the file would have been overwritten by 10 "extended "s, because the file pointer would have started at zero. The file length would have stayed the same.

The file length is the maximum value you can use with **POINT,** so if you want to extend a file, even with blank records, you must **POINT** to the end and use **PRINT** # to increase the file size.

You may wonder why line 140 does not read: **140 POINT #4,FN L(4).** For technical reasons you cannot combine a Beta DOS or G+DOS command with one of the new functions in a single statement. Use a temporary variable to avoid this, like **"length"** in the example above. Why do I keep using **"L"** instead of **"l"?** Because it saves confusion with **"1"** in listings.

Now let's read the entire file to check that all is as we expect:

```
200 CLOSE #*4
210 OPEN #4;dl"test"RND
220 INPUT #4;a$
230 PRINT a$
240 GO TO 220
```

This finishes with an **"END of file"** report, which can be a problem when you are writing a real program. It is possible to end the file with a **"rogue value"** like **"zzz"** and use a line like: 240 IF a$<>"zzz" THEN GO TO 220 but this is not very convenient. Beta DOS provides a new function to tell you when the end of the file has been reached so that you can avoid further **INPUTS** - see below.

FN E - END-Of File Function

To use this function you need a line like this somewhere in your program:

```
190 DEF FN e(x)=USR 8
```

The function returns a zero if the last character in the specified stream has not been read yet, or a one if it has. The example above could make use of a modified line 240:

```
240 IF FN e(4)=0 THEN GO TO 220
```

Unfortunately you can only use these three new functions on **OPENTYPE** files which have been **OPENed** using **RND.**

DATA PACKING IN RANDOM-ACCESS FILES

Our "test" file is fairly small and simple. A real example would probably use longer records, with different parts of the records dedicated to particular purposes. (These areas are called **"fields"**.) For example, if each record is created by **PRINTing** a 100-character string to the file, you might place data in a record as shown below. (This partial program is not on your disc.)

```
100 DIM d$(100)
110 INPUT "Author? ";a$
120 LET d$( TO 40)=a$
130 INPUT "Title? ";t$
140 LET d$(41 TO 96)=t$
150 INPUT "Year of publication? ";y$
160 LET d$(97 TO 100)=y$
170 PRINT #something;d$
```

(You can also do a direct INPUT d$(TO 40) but the method above is handier when you come to add error—trapping.)

Having read such a record from a file, you could display the information like this;

```
500 PRINT "Author:";d$( TO 40)
510 PRINT "Title:"id$(41 TO 96)
520 PRINT "Year:";d$(97 TO 100)
```

Some data can have an annoying number of fields – an address, For example, can include street, district, town, county and Postcode. If you reserve enough space for the maximum field Length assumed possible, you waste LOTS of disc space. An Alternative approach is to stay with a fixed-length for each complete record, but handle the contents a bit more flexibly. For sample, we could store a number of variable-length items per record as shown in the incomplete program section below:

```
600 DIM d$(100)
610 LET t$=""
620 INPUT a$
630 IF a$="" THEN GO TO 660
640 LET t$=t$+a$+CHR$ 13
650 GO TO 620
660 LET d$=t$
670 PRINT #4;d$
680 REM rest of program
```

The items are separated by carriage returns, which means we need multiple **INPUTS** to read each complete record, as shown by the incomplete input routine below:

```
900 LET ptr=FN p(4)
910 INPUT #4;a$
920 PRINT a$
930 IF FN p(4)<ptr+100 THEN GO TO 910
```

The use of **FN P** as shown allows a variable number of sub-strings to be stored in each record. The data could be built up into a single string, instead of being printed, by something like **LET t$=t$+a$+CHR$ 13**, lf you Preferred. You could also have stored the data on disc using something other than **CHR$ 13** as a separator. Any character with a code between **128** and **255** will do, since it probably will not occur in the data itself - e.g. **CHR$ 128.** This will allow you to read the record with a single **INPUT**, but require you to search for the separators in order to distinguish individual items.

STORING NUMBERS

Let's assume you want to store numeric data in a file. Sometimes you can simply **PRINT** the number, as in: **PRINT #5;x.** However you will have more control over exactly what part of the file is used if you do something likes **LET d$(10 TO 12)=STR$ x.** The details of the best method to use will depend very much on the range and precision of numerical values you want to store and on how keen you are to save space. For example, if x is a whole number between 0 and 255 you can use e.g.: **LET d$(65)=CHR$ x.** (**LET x=CODE d$(65)** is the reverse.) You can limit a value to a fixed number of decimal places using e.g. **LET x=INT (x*100)/100** before storage (this limits to two decimal places).

VARIABLE-LENGTH RECORDS

Sometimes fixed-length records are unsuitable because the data you are dealing with is so variable in size that too much disc space would be wasted. Sometimes variable-length records may be just simpler to program, particularly if you rarely or never want to update a record (which is tricky, expecially if the new data is longer). Also, you may want to read data from existing serial files made up of variable-length records. Once you are used to random-access you may find the delay associated with reading a record part-way through such a file (using multiple INPUTS) rather Irritating. Unfortunately, the variable record length means that it is not possible to use the normal form of POINT to access a given record; e.g.:

POINT #stream,(record number-l)* record length

There are devious ways round this, like keeping a file of fixed-length data giving pointer values to each variable-length record in another file, or a later part of the same file. But let's keep things simple, and exploit another feature of **POINT** instead. Something like:

POINT #5, OVER 10

will start from wherever the current file pointer is, and pass over 10 carriage returns before setting a new pointer position. So to point to the 2000th. record in a file, we could use:

POINT #5,0: POINT #5, OVER 1999

This will point to the start of the file, and then pass **OVER** 1999 carriage returns. Although **POINT** will have to read the first 1999 records in order to do this, it reads at about 22.5K per second, so that for many files the time required is insignificant. Besides, if you know what record you have just INPUT, you often do not have to start at the beginning of the file again. For example, if you have **INPUTed** record number 200B, and want to **INPUT** record 2100 next, **POINT #5, OVER 99** will work. It is even possible to re-define the character **POINT OVER "passes over"** as the program runs, using **POKE @125, (character code),** so you can separate records by one character (say, **CHR$ 128**) and fields by another, and use **POINT OVER** to find both the record and the field you want, very quickly. To illustrate this, the listing below creates a file of 1000 records, each terminated by **CHR$ 128** and containing 6 random-length fields ending in **CHR$ 13**. The semi-colons at the end of lines 60 and 80 prevent carriage returns being sent to the file. You can load the program from the Beta DOS disc using **LOAD dl"prog3"**. RUN to create the file. This will take some time, as the file will be about 150K long. Go for a tea or coffee break!

```
10 CLOSE #*4
20 OPEN #4;dl "varifile" RND
30 FOR r=l TO 1000
40 PRINT AT 10,10;r
50 FOR f=1 TO 6
60 PRINT #4;"Record;";r;" Fields";f;
70 PRINT #4;" abcdefghijklrnn"( TO RND*14)
80 NEXT f: PRINT #4;CHR« 128;
90 NEXT r
100 CLOSE #*4
110 STOP
```

When you get back, the program below will let you read any desired field and record from the files:

```
200 CLOSE #*4
210 OPEN #4;dl"varifile"RND
220 POINT #4,0
230 INPUT "Record? ";r
240 IF r=0 THEN STOP
250 INPUT "Field? ";f
260 POKE @125,128
270 IF r<>1 THEN POINT #4; OVER r-1
280 POKE @125,13
290 IF f<>1 THEN POINT #4; OVER f-1
300 INPUT #4;r$: PRINT r$
310 60 TO 220
```

OPENING MULTIPLE FILES

Our examples have only dealt with one file being open at a time, but it is quite possible to have many files open at once in random-access mode. However, if the files you are using involve writing to a new file, or extending an old one, then all those files must be on the same disc and disc drive. (Other files can use the second drive.) This is similar to the way in which G+DOS requires all **OUT** files to be on the same drive.

Open files take almost 800 bytes of **RAM** so you may run out of memory when you try to do an **OPEN**.

MOVE and Random—Access Files

It is possible to **MOVE** a file to a stream **OPENed** to a random—access file. Since **MOVE** uses the equivalent of **PRINT** # to transfer data, the results are easy to predict. If you have just **OPENed** the random-access file, the file pointer will be at zero and the **MQVEd** data will overwrite the existing file until it has all been overwritten and the file begins to extend. If you use POINT and the file length function to set the file pointer to the file end, the data will extend the file without overwriting any of the existing data.

SOME DIFFERENCES FROM G+DOS

There is a problem with G+DOS`s save-a-sector routine; if the disc has stopped, and the destination sector requires the disc head to move to a new track, the sector may be written before the disc has reached full speed. This can corrupt the sector being written, and often the next one on the same track. In mild cases the sector can still be read, but with difficulty. Often the only cure is to re-FORMAT the disc. The problem can occur during **SAVE @** using G+DOS - Beta DOS corrects this. It can also occur when a serial output file is being written, and Beta DOS does not correct this for files **OPENed** using **OUT**. However, a file **OPENed** using **RND** can be used as a serial output file and does not suffer from the fault.

If you use **RND** instead of **OUT**:

Writing will be much more reliable.

There will be no error message if the file exists, and you might overwrite a file without wanting to. You can check for this possibility using the file Length function, **FN L**, which will be zero if you have just **OPENed** a new file.

If you use **RND** instead of **Ins**

There will be no error message if the file does not exist, until you try to INPUT from it. (This will give an **"END of file"** report.)

There will be no error message if you write to the file.

The most common source of accidental attempts to write to an input file is the **INPUT** command. The examples below will produce a **"Writing a READ file"** report with a file **OPENed** with **IN**, whether you are using Beta DOS or G+DOS:

```
INPUT #4;"address? ";a$
INPUT #4;a$,b$
```

This is because the prompt, and less obviously, the print comma, both try to write to stream 4. When you are using a file **OPENed** using **RND**, Beta DOS presumes you do not really want to write to a disc file during an **INPUT**, and ignores any such attempts without doing any writing.

Using G+DOS it is possible to part-**OPEN** a disc file if you press **Break** or there is a disc error during **OPEN**. This can cause a reset if you then try to **CLOSE**, **OPEN** or **CLEAR #**. This should no longer be possible, although you may have to use **CLEAR #** to reclaim the memory used by **OPEN**.

Rarely, G+DOS can lock-up the Spectrum due to interrupts being disabled on return from a disc operation. This means that the keyboard is ignored. Beta DOS has enough sections of G+DOS attached to it to have the same problem, but it allows you to escape from this situation by pressing the disc interface button and then the zero key (a method first described in **FORMAT** magazine)

PEEKING THE PLUS D`S MEMORY

Beta DOS provides two extra functions which can be useful, particularly if you are interested in the "techie" side of things. **FN S** provides a **PEEK** of "Shadow" **ROM** and **RAM** i.e. the **ROM** and **ROM** in the PLUS D, which is normally switched off unless a disc operation is in progress. The little program below will let you examine the PLUS D-s **ROM** at addresses **0-8191**, its RAM at **8192-16383**, and the normal Spectrum RAM at 16384-65535.

```
10 DEF FN s(x)=USR 8
20 INPUT "address?";a
30 PRINT a;" ";FN s(a)
40 LET a=a+1: GO TO 30
```

(For a more complex example of the use of **FN S**, look at the **BACKUP** program on your Beta DOS disc.) This function relates in a very simple way to **POKE @**, which allows **POKEs** to be made to the PLUS D RAM. POKE @0,x **POKEs** a value into address **8192** - the first ("zeroth") byte of shadow RAM. You can read the value you **POKEd** with **PRINT FN s(8192)**, or, for an even easier method use a second function, **FN A**, which simply calls **FN S** with an offset;

```
1 DEF FN s(x)=USR 8: DEF FN a(x)=FN s(x+8192)
```

Then **PRINT FN a(address)** will refer to the same byte as **POKE @address**,x.

SYSTEM VARIABLES

You can use **POKE @** and **FN A** to change or monitor the state of Beta DOS using the following system variables.

124	**DTKS**	**The number of tracks in the catalogue. Set by SAVE,LOAD, CAT, ERASE. Will be 4 for a disc with a normal 80-file catalogue.**
125	**DELIM**	**Character code used by POINT as a record terminator. Usually 13, for a CHR$ 13.**
126	**COLS**	**Number of columns in a simple CAT. Usually 3.**
127	**COMP**	**Compression flag. If "squash" has been loaded, this will be 1, signalling "compress Snapshots". POKE @127,0 to turn off compression.**

4480	**PTRLET**	**Usually 80 (P) Letter for Pointer function.**
4481	**EOFLET**	**Usually 69 (E) Letter for End of File function.**
4482	**SPKLET**	**Usually 83 (S) Letter for Shadow PEEK function.**
4483	**LENLET**	**Usually 7& (L) Letter for Length of File function.**

Locations **4480-4483** contain the character codes for **,FNs P, E, S** and **L.** If you have programs that already use these function letters, you can redefine the letters used by the Beta DOS functions by POKEing new character codes here. For example, **POKE @4480,81** to allow use of **DEF FN Q(x)»USR 8** and **FN Q(x)** in place of **FN P(x).**

4484	**SRTFG**	**Usually 1, which turns on alphabetic sorting of simple CATs. POKE @4484,0 to turn this off.**
7652	**NBOT**	**Beta/G+DOS. POKE @7652,0; NEW.- RUN to re-boot DOS**
7663	**SAMCNT**	**Beta/G+DOS "files open" counter. Zeroed by. CLEAR #.**

To prepare a copy of Beta DOS which will be already **POKEd** when you load it, instead of: **POKE Saddr,value** use this:

```
10 CLEAR 29999: LOAD dl"+sysBeta"CODE 30000
20 POKE addr+30000, value: SAVE dl "+sysBeta"CODE 30000,6850
```

BACKUP UTILITY

The BACKUP utility on your disc is a Basic program that uses **LOAD @** and **SAVE @** to copy all the files on a disc in 38K or larger chunks. The new Shadow **PEEK** function **(FN S)** is used to read the PLUS D's sector allocation map for the source disc so that unused tracks are not normally copied. If you are using a 128K Spectrum in 128K mode the RAM disc is used as a temporary store to minimise disc swapping. The program shows you how many swaps are needed unless it can do the Backup in one go. If you are lucky enough to have two disc drives, alter line 40 of the program to **LET dd=2**. The program will then copy from drive 1 to drive 2 without any disc swaps.

SQUASH - COMPRESSION UTILITY FOR 4BK and 128K SNAPSHOTS

This utility is provided as a separate **EXECUTE** file because there isn't enough space in the PLUS D's RAM to include it as an integral part of Beta DOS. To use the utility, simply **LOAD dl"squash"**x and the program will load and integrate itself with either Beta DOS or G+DOS in the PLUS D's RAM. Beta DOS will lose the ability to sort file names in a **CAT** alphabetically, and the new functions will no longer work, because they have been over-written. (Don't worry - **CAT** still works, it just doesn't sort, and use of the functions will give a **"FN without DEF FN"** report - the program will not crash.) When **"squash"** is used with G+DOS, no facilities will be lost, and the modified G+DOS can be saved using e.g. **SAVE dl"+sys sq' CODE 8192,6656** if desired.

When **"squash"** has been loaded, any 48K or 128K snapshot you make will be compressed before saving. With 48K snapshots there will be brief screen corruption as compression occurs. The files created in this way appear as normal **SNAPSHOTS** in the directory, but of course the size will be less than the normal 97 or 258 sectors.

The degree of compression will vary a lot according to the state of the computer when the snapshot was made. If you have just turned the machine on, and the screen is fairly clear, very large compressions are possible because most of memory is filled with repeated zeros. For a 128K snapshot, this may still be true even after a game has been loaded, but compressions possible with a professional 48K game are usually more modest - perhaps to 80% of the original size. Still, you should be able to fit lots more snapshots on a disc! Best results are achieved when the memory is as unused as possible, so it is a good idea to reset the machine before starting. You can do that with a Reset button, if you have one, or use a facility provided by **"squash"** - press the interface button and then **SPACE**.

When you come to reload a compressed snapshot you must use a DOS which has had **"squash"** added to it. Both compressed and uncompressed snapshots can then be loaded, so you can load a series of normal snapshots and re-SAVE them in compressed form. You will see brief screen corruption as compressed 4BK snapshots are loaded.

It is possible that you might want to load compressed snapshots and then save them as normal uncompressed snapshots. To do this, **POKE @127,0**. Normal or compressed snapshots can then be loaded but all saved snapshots will be normal . Compression can be turned back on by **POKE @127,1**.

Undocumented Commands

CAT 1””””” Will return the amount of disk space left on drive 1. [4 “s]

Useful Poke @`s

Poke @0,n [n=0 to 7] ; Default-. 7.
Flash border when loading
0 means NO flash at all.

POKE 03,n Step rate for disk drive. [Default 36ms]

POKE @299,64: POKE @2330.64
Resets Snapshot to 16K ,33 Sectors.

POKE @299,192: POKE @2330,192
Change above to 48K Snapshot.

POKE @3780,((8*paper)+ink)
Resets screen colour for CLS #.

POKE @6000,xxxx DOS Checksum.
poke 10 and you fool the DOS
into thinking its not loaded.

POKE @7652,0 Resets the DOS system

POKE @23755-8192,65535
Merge proof loader. First line must be:-
1 REM

POKE @23757-8192,65535
Before you save makes first line 64K
long, gives Out of Memory error.

PULS D MEMORY MAP:

0 - 8191 - ROM.
8192 - 16383 - RAM.

Beta Dos Fixer

```
10    REM     BETADOS BUG FIXES
15    REM    (INCLUDING SNAPFIX)
20    REM    (PD) By Miles Kinloch
22    REM     from FORMAT Vol 9.9
30    REM
32    REM Typed by SPT 14-06-04
40    CLEAR 4e4: RESTORE : LET check=1: LET c1=0: LET
      c2=0: LET c3=0: FOR a=40001 TO 40065: READ d:
      LET c1=c1+d: POKE a,d: NEXT a: DATA
      33,0,160,17,1,160,1,193,26,54,0,237,176,201,221,
      33,0,0,33,0,160,1,97,13,94,35,86,35,221,25,11,12
      0,177,32,245,221,229,193,201,221,33,0,0,17,0,160
      ,1,194,26,26,183,32,2,221,25,19,11,120,17
      7,32,244,221,229,193,201: REM Routine to perform
      initial check of data
42    PRINT "Confirming data - please wait..."
45    IF c1<>6330 THEN PRINT "CHECK DATA - ERROR IN
      LINE 40!": STOP
50    RANDOMIZE USR 40001: GO SUB 70: LET c2=USR
      40015: IF c2<>40484 THEN PRINT "CHECK DATA -
      ERROR IN BYTES!": STOP
52    LET c3=USR 40040: IF c3<>48180 THEN PRINT "CHECK
      DATA - ERROR IN ADDRESS!":STOP
55    PRINT "Put a disk with a PREVIOUSLY UN-MODIFIED
      Betados system file in Drive 1,then press any
      key.": PAUSE 0
60    SAVE d1"+sys Beta"CODE 40960
65    CLS : PRINT "Please wait....": RESTORE 70: LET
      check=0:
70    FOR a=44149 TO 44160: READ d: POKE a,d: NEXT a:
      DATA 191,40,9,254,223,40,5,205,236,54,32,4: POKE
      44156,254: POKE 44157,165: POKE 44158,0: REM
      Fixes OPEN #command statement end
80    FOR a=46403 TO 46410: READ d: POKE a,d: NEXT a:
      DATA 215,130,28,247,200,215,153,30: REM Fixes
      FORMAT dn,n command to work on drive specified
90    FOR a=45175 TO 45236: READ d: POKE a,d: NEXT a:
      DATA
      78,111,46,32,111,102,32,70,114,101,101,32,75,45,
      66,121,116,101,115,32,61,160,205,215,34,210,124,
      4,205,220,2,195,72,24,237,115,102,32,205,178,50,
      24,7,237,115,102,32,205,173,50,195,220,2,205,2,7
      ,58,16,62,195,181,36: REM Fixes command codes
      67, 68 and 69, also sorts SAVE @/Load@ error bug
100   POKE 41726,172: POKE 41727,48: POKE 45729,166:
      POKE 45730,48: POKE 45737,157: POKE 45738,48:
      POKE 45446,220: POKE 45447,2: POKE 43178,153:
      POKE 43179,48: POKE 43181,141: POKE 43182,48:
      POKE 43267,162: POKE 43268,48: POKE 43270,141:
      POKE 43271,48: POKE 45749,246: POKE 45750,50:
      POKE 45789,42: POKE 45790,210: POKE 45791,58:
      POKE 45798,204: POKE 45799,228: POKE 45800,12:
      POKE 47627,195: POKE 47628,121: POKE 47629,49:
      REM Miscellaneous POKEs to call new patches etc.
110   FOR a=45812 TO 45818: READ d: POKE a,d: NEXT a:
      DATA 167,201,211,239,195,246,55: REM Clears
      carry flag after successful SAVE@/LOAD@ command
      spin proper drive before saving
```

```
120   FOR a=45250 TO 45318: READ d: POKE a,d:
      NEXT a: DATA
      22,1,7,73,110,115,101,114,116,32,83,79,85,
      82,67,69,32,100,105,115,227,205,138,23,22,1,7,73
      ,110,115,101,114,116,32,84,65,82,71,69,84,32,100
      ,105,115,227,58,26,62,184,202,212,35,219,235
      ,50,210,35,58,211,35,211,235,58,210,35,50,211,35
      ,201: POKE 45246,24: FOR a=41935 TO 41939: READ
      d: POKE a,d: NEXT a: DATA 195,239,48,0,0: REM
      Increases speed and reliability of SAVE
      ...TO.... when copying between 2 drives
130   FOR a=44916 TO 44949: POKE a-4,PEEK a: NEXT a:
      FOR a=44946 TO 44963: READ d: POKE a,d: NEXT a:
      DATA 205,79,49,195,2,7,205,135,47,195,101,52,
      205,152,47,195,121,40: FOR a=44971 TO 45006:
      POKE a-7,PEEK a: NEXT a: FOR a=45000 TO 45011:
      READ d: POKE a,d: NEXT a: DATA
      195,79,49,235,237,91,197,58,58,79,62,233: FOR
      a=41643 TO 41686: READ d: POKE a,d: NEXT a: DATA
      217,229,217,33,197,34,229,237,115,102,32,195,203
      ,47,214,24,254,21,210,55,49,17,222,34,24,212,229
      ,205,215,34,225,217,225,217,205,142,22,195,80,0,
      225,217,225,217: REM Preserve HL'
140   POKE 41621,35: POKE 41110,135: POKE 41694,135:
      POKE 41696,152: POKE 41698,158: POKE 41702,164:
      POKE 41710,170: POKE 41714,176: POKE 41722,182:
      POKE 41734,211: POKE 44965,196: POKE 44971,135:
      POKE 44977,196: POKE 44983,135: REM (Alters the
      call to relocated routines)
150   POKE 41704,217: POKE 41705,52: REM Alters
      command code 56 to be able to cope with disks
      formatted for more than 80 files
160   FOR a=43828 TO 43848: READ d: POKE a,d: NEXT a:
      DATA
      205,79,49,205,218,43,48,13,24,93,205,72,49,205,2
      ,7,205,76,14,24,82: FOR a=42777 TO 42785: READ
      d: POKEa,d: NEXT a: DATA
      24,64,241,24,81,254,88,32,93: POKE 42701,77:
      POKE 42770,11: POKE 42707,62: POKE 42708,43:
      POKE 43803,10: REM Creates command LOAD d1 ect.
      To change current drive. Also fixes form of MOVE
      command: MOVE #s to dn"name"
170   POKE 46391,245: POKE 46392,205: POKE 46393,177:
      POKE 46394,36: POKE 46395,241: REM Fixes FORMAT
      d* command so syntax is accepted )in
      conjunction with line180)
180   FOR a=42318 TO 42328: READ d: POKE a,d: NEXT a:
      DATA 195,124,4,58,58,92,254,14,192,225,201: FOR
      a=42154 TO 42164: READ d: POKE a,d: NEXT a: DATA
      221,33,108,253,195,80,0,175,195,36,51: POKE
      43147,205: POKE 43148,81: POKE 43149,37: POKE
      43150,0: POKE 47636,170: POKE 47637,36: REM
      Keeps IX correct after FN calls for
      better compatibility with 128 BASIC, sorts error
      message with SAVE " ", fixes problems with
      capital D when saving or erasing. Also part of
      ode for fixing FORMAT d* syntax bug.
185   FOR a=45396 TO 45439: READ d: POKE a,d:
      NEXT a: DATA
      205,138,23,127,32,77,71,84,32,38,32,66,101,116,9
      7,83,111,102,116,32,40,49,57,57,48,41,32,66,68,7
```

```
      9,83,32,49,46,51,13,141,62,201,50,155,32,24,212:
      REM Relocate boot message in buffer area
190   FOR a=45078 TO 45093: READ d: POKE a,d: NEXT a:
      DATA
      205,79,49,195,132,5,254,160,200,254,83,200,195,1
      98,55,225: FOR a=41213 TO 41217: POKE a,0: NEXT
      a: POKE 42606,37: POKE 42607,48: POKE 47043,195:
      POKE 47044,28: POKE 47045,48: POKE 46828,167:
      POKE 46829,195: POKE 46830,142: POKE 46831,22:
      POKE 47063,236: POKE 47064,54: REM Fixes
      problems with printing to a file opened RND
191   REM Lines 192-197 SNAPFIX
192   FOR a=45250 TO 45294: READ d: POKE a,d: NEXT a:
      DATA 73,110,115,101,114,116
      ,32,79,82,73,71,32,100,105,115,227,205,138,23,73
      ,110,115,101,114,116,32,84,65,82
      ,71,69,84,32,100,105,115,227,245,205,56,46,241,1
      95,21,33
193   FOR a=44593 TO 44610: READ d: POKE a,d: NEXT a:
      DATA 24,40,205,72,49,24,14,
      42,254,63,62,103,195,234,45,205,185,46
194   POKE 45246,19: POKE 44576,18: POKE 44580,14:
      POKE 41211,64: POKE 41206,195:
      POKE 41122,231: POKE 41123,48: POKE 44736,195
195   FOR a=44739 TO 44747: READ d: POKE a,d: NEXT a:
      DATA 205,185,46,33,56,0,34,141,92
196   FOR a=44519 TO 44530: READ d: POKE a,d: NEXT a:
      DATA 195,125,15,35,35,35,35,190,192,195,112,40
197   FOR a=43120 TO 43127: READ d: POKE a,d: NEXT a:
      DATA 35,175,190,192,43,34,254,63
200   CLS : IF check THEN RETURN : REM return here
      from initial data check
210   INPUT "Is your spectrum a 128k machine (y/n)? ";
      LINE a$: IF a$="N" OR a$="n" THEN GO TO 270
220   IF a$<>"Y" AND a$<>"y" THEN GO TO 210
230   INPUT "Is your spectrum a 2a machine (y/n)?
      "'"Answer 'n' if it has been fitted with a 2a
      ROM chip by installing B.G. Services conversion
      kit."'' LINE a$:
      IF a$="Y" OR a$="y" THEN POKE 42156,152: POKE
      45088,67: GO TO 270
240   IF a$<>"N" AND a$<>"n" THEN GO TO 230
250   INPUT "Is your spectrum an original Sinclair
      machine i.e. the type with the
      black case (y/n)?"'"Note - some early 2's used
      the old Sinclair casings, so answer 'n' it has
      an Amstrad ROM."'' LINE a$: IF a$="Y" OR a$="y"
      THEN POKE 45088,52: GO TO 270
260   IF a$<>"N" AND a$<>"n" THEN GO TO 250
270   PRINT "Press any key to resave Beta Dos.":
      PAUSE 0: CLS
280   SAVE d1"+sys Beta" CODE 40960,6850
290   STOP
9999  SAVE d1"BetaFix"
```

For more information about Beta Dos bugs fixer & bugs Fixed see:

Format Magazine Vol 9.8 – 1996 thanks to Bob Brenchley and
Format Magazine Vol 9.9 – 1996 Miles Kinloch
Format Magazine Vol 9.10 – 1996