



DOSplus 4 .6

for the

DRAGON  
COMPUTER



# CONTENTS

1.0	Introduction	3
2.0	Getting Started	3
2.1	Inserting a Disk	3
2.2	Selecting a Drive	4
2.3	Initialising a Disk	4
2.4	DOSplus Workspace and Graphics Ram	4
2.5	DOSplus Disable	4
2.6	Keyboard Repeat	5
3.0	Data Storage	5
3.1	Reading and Writing Disk Sectors	5
3.2	Filenames	6
3.3	Listing Disk Contents	6
3.4	Disk Free Space	6
3.5	File Lengths	7
3.6	Copying Files	7
3.7	Changing a File Name	7
3.8	Deleting Files	8
3.9	Delete Protecting Files	8
4.0	Program Files	8
4.1	Storing a BASIC Program on Disk	8
4.2	Loading and Running a BASIC Program	9
4.3	Calling Programs in Sequence	9
4.4	Combining BASIC Programs	10
4.5	Binary Files	11
4.6	File Backup	11
5.0	Data Files	11
5.1	Opening Files	12
5.2	Creating Files	12
5.3	Closing Files	12
5.4	File Input and Output	13
5.5	File Read Pointer Manipulation	14
5.6	'End of File' Processing	14
5.7	FREAD and FWRITE Extensions	14
5.8	PRINT #n modes	16
5.9	Machine Code Character I/O	16
6.0	Miscellaneous Features	16
6.1	Selective RESTORE	16
6.2	Text Command Files	16
6.3	Write Verification	17
6.4	Disk Backup	17
6.5	Pausing a Program	17
6.6	Memory Allocation	18
6.7	Error Trapping	18
6.8	Warning 'Bell'	18
6.9	Data Exchange	19
6.10	Automatic Line Numbering	19
6.11	Naming and Numbering Disks	20
6.12	Booting an Operating System	20
Appendix 1:	DOSplus Extensions to BASIC	21
Appendix 2:	Facilities for Assembler Programmers	39
Appendix 3:	DOSplus Hooks and Workspace	47
Appendix 4:	DOSplus File Header Format, Directory Sector and Record Format	51
Appendix 5:	Error Codes	55
Appendix 6:	Command Summary	59





## 1.0 Introduction

Welcome to the world of **DOSplus 4** - a DRAGONDOS compatible Disk Operating System (DOS). This means that, when running programs written for DRAGONDOS or issuing commands in standard DRAGONDOS format, it should not be possible to detect which DOS is actually being used. There are two exceptions to this: firstly directory backup is 'on demand' and not automatic and secondly directory listings scroll slowly to the screen. To whet your appetite for what follows, amongst the additional features you will find are:

- Autoboot from drive 1 at power on or any 'cold' reset
- Slow directory scroll to the screen
- Keyboard repeat for both DRAGON 32 and 64 (not DELTA on D32)
- Disk to disk file copy with a single drive
- 'On demand' character input/output for disk files
- DOS disable capability

plus

- Error checking and detection which is difficult to bypass.

DOSplus can support up to four Shugart SA400 compatible drives (3", 3½" or 5¼") in any mix of single or double sided disks and with 40 or 80 tracks, but the standard drive is assumed to be a single sided, 40 track unit. Note that 8" disks and, except for DELTA SD, single density formats are **not** supported, though most cartridges can operate in single density mode.

Following the descriptive text, the appendices provide a quick reference to the facilities available for BASIC and assembler users. Appendix 2 details all the 'legal' entries into DOSplus, which provide all the facilities needed to handle disks.

## 2.0 Getting Started

Before applying power to the DRAGON and the disk drive, you should connect the units together as described in your hardware manual. When all is correct, turn on the power to the peripherals and then to the DRAGON.

The television screen (or monitor) should now display the DOSplus message followed by the usual DRAGON and Microsoft copyright messages and a request for a key press. Press any key (except Y) and the OK message and the cursor should appear. If a disk is inserted in drive 1 and the Y key is pressed, DOSplus will attempt to load an operating system (such as FLEX or OS9).

## 2.1 Inserting a Disk

Insert the disk into the drive aperture with the disk heads slot towards the drive and the label uppermost. Secure the disk and engage the head(s) by turning the locking lever or closing the 'door'. To remove the disk, turn the lever back to the open position, press the door release or eject button, as appropriate.

## 2.2 Selecting a Drive

DOSplus is capable of handling up to four drives connected to the DRAGON. If more than one drive is connected, it is important that the commands use the correct drive. The **DRIVE** command may be used to change the default drive number (mnemonic **DEFD**, location 1546 or \$60A) in the workspace. When a drive number is not explicitly specified in a command, the current value of **DEFD** is used. At power up or a 'cold' reset, **DEFD** is set to 1. The command

**DRIVE n**

alters the value of **DEFD** to **n** (**n** = 1 to 4).

## 2.3 Initialising a Disk

When a disk is initialised, it is divided into concentric rings or 'tracks', with each track divided into a number of sectors of 256 bytes. There are 18 sectors per track on a single sided disk and 36 sectors (18 per side) on a double sided disk (10 and 20 sectors respectively for a single density disk). This formatting must be done before a new disk can be used and is necessary to allow disk housekeeping to take place. Initialising a disk is done by invoking the **DSKINIT** command:

**DSKINIT drive,sides,tracks**

If the disk has previously been formatted, DOSplus will ask for confirmation with '**SURE?**'. Type '**Y**' to format the disk, **<BREAK>** to abort without error or any other key to abort with a **?PR ERROR**. The parameter **drive** is the drive number (which defaults to **DEFD**, as set by the **DRIVE** command), **sides** is the number of sides to be formatted (1 or 2, default is 1), and **tracks** is the number of tracks per side (usually 40 or 80, with 40 as default). For the standard **DRAGON** disk drive (single sided, 40 track) you need only type '**DSKINIT**'. The disk drive will operate for a few minutes displaying an indication of the track being written (formatted) or read (checked), after which '**OK**' should appear. If an error occurs, remove and re-insert the disk and try again. If the disk repeatedly fails to format, the disk may be faulty, though dirty heads should also be considered, especially if read/write errors have previously occurred.

If other than the defaults are required then, for example,

**DSKINIT 2,2,80**

will format a disk as double sided 80 track in drive 2.

If, for any reason, it is required to format a disk with other than 40 or 80 tracks, DOSplus will request confirmation of the number before continuing. As before, answer '**Y**' to accept, **<BREAK>** to abort cleanly or any other key to abort with an error message.

Note that the drive must be capable of handling the format specified, or damage to the drive may result. Note also that a disk is normally only formatted the first time it is used, as formatting the disk again will over-write all information on the disk. However, this can also be used as a quick method of deleting **all** the files on a disk.

## 2.4 DOS Workspace and Graphics RAM

With the DOS cartridge installed, DOSplus workspace and file buffers occupy the memory from 1536 (\$600) to 3071 (\$BFF). This means that the first page of the graphics RAM now starts at address 3072 (\$C00) instead of 1536 (\$600). This can be a problem with some programs designed to be run in a **DRAGON** without disk drives, since if the program writes to the area used by DOSplus for workspace, the disk operating system data can be corrupted. To recover from the situation, **POKE 113,0** and press the reset button or switch the computer off and on again.

## 2.5 DOSplus Disable

For those occasions where it is necessary to run without a DOS capability (for example: running cassette based games which write to the DOSplus workspace area as mentioned in 2.4 above), it is no longer necessary to remove the cartridge. At any time input is accepted, alter the 'secondary reset vector' by:

**POKE 115,PEEK(115)-10**

and press the reset button. This will perform a partial cold start, retaining the top of memory and string space pointers, but setting the graphics base to 1536 (\$600) and clearing any **BASIC** program. DOSplus can be re-enabled by performing a full cold start (**POKE 113,0** and press reset) or **POKE 235,0 : EXEC 49154**.



## 2.6 Keyboard Repeat

For full compatibility with DRAGONDOS, DOSplus powers on in 'standard' DRAGON keyboard mode. If you wish to switch between the 'standard' keyboard mode of operation and roll-over or repeat, this is done by setting memory location 270 (\$10E) to one of the following values:-

DOSplus 4		DOSplus DELTA
137 (\$89)	Keyboard repeat and roll-over	N/A
138 (\$8A)	Keyboard repeat and roll-over	156 (\$9C)
180 (\$B4)	Roll-over, no repeat	180 (\$B4)
186 (\$BA)	'Standard' DRAGON keyboard operation	186 (\$BA)

For some programs, the repeat and/or roll-over options are not compatible with the built-in keyboard repeat and the 'standard' mode should be used.

## 3.0 Data Storage

There are two ways of storing data on a disk – the hard way, where the user has to control the allocation of space, and the easy way, where DOSplus does the housekeeping. In the latter case, DOSplus allocates space to files (blocks of associated data, programs etc. – see 3.2 et seq) and writes the details to a reserved area of the disk, known as the **directory**. A **filename** – that is, a **rootname** and **extension** – to be associated with the data is also written to the directory record. This filename is used by various commands to access the data through DOSplus.

As DOSplus needs the disk format to correctly access files on a disk, it is important to close all files on a disk before it is removed from the drive and a new disk inserted. This will not only ensure that the directory of the disk is correct but will force the disk format to be read from the disk the next time the directory is accessed for file information.

One point to note when using disk drives with 40/80 track switching is that if the drive setting is not correct for the disk, any disk access will usually result in a ?RF ERROR (the exception is an access of track zero).

## 3.1 Reading and Writing Disk Sectors

For those who like to do things the hard way, each 256 byte sector created when a disk is initialised can be accessed through the **SREAD** and **SWRITE** commands. These commands allow data to be manipulated without using the DOSplus file structure. Note, however, that these commands will not generate a ?TR ERROR if the disk format has not been read.

To read data from a sector, the command

**SREAD drive,track,sector,A\$,B\$**

is used, where **drive** is the disk drive number, **track** is the track number and **sector** is the sector number. **A\$** and **B\$** are two string variables which will be used to hold the data read from disk. Each of these strings will contain exactly 128 bytes (characters) of data, which may be CHR\$(0) padding if the strings used to write the sector were less than 128 bytes long (see **SWRITE**). Note that **CLEAR** must be used to get sufficient string space to hold the 256 bytes read from disk.

To write data to a sector, it is assembled as two strings and the command

**SWRITE drive,track,sector,A\$,B\$**

is used, where **A\$** and **B\$** contain the data to be written and all the other parameters are the same as for **SREAD** above. In this example, the string **A\$** will be written to the first half of the sector and **B\$** to the second half. If a string is less than 128 bytes, CHR\$(0) characters will be used to fill the remainder of the 128 bytes of that half sector. If a string is longer than 128 characters, only the first 128 are used.

## 3.2 Filenames

For those who prefer to let the computer do the work, DOSplus can be used to control disk space allocation. This is done by reference to a file, using its filename. The full format of a filename takes one of the four forms:

<b>drive:rootname.extension</b>	or	<b>drive:rootname/extension</b>
or <b>rootname.extension:drive</b>	or	<b>rootname/extension:drive</b>

where **rootname** is a name of up to eight 'name' characters (that is: alpha, numeric or '-' characters), normally starting with a letter, and **extension** is a code of up to three 'name' characters used to distinguish files with the same rootname. The following three letter codes are used as defaults by certain commands to identify files of BASIC, machine code, etc. (See the individual command descriptions for details of when this occurs).

BAK	-	Backup file
BAS	-	BASIC program
BIN	-	Binary file, for example a machine code program
CMD	-	Command text file
DAT	-	Data file

While there is generally no restriction on which form of file reference you use – for example, 1:SCREEN.GRA and SCREEN.GRA:1 refer to the same file – there is one special case. If you refer to a file with a single character rootname and give a drive number but do not provide an extension, the drive number **must** follow the name (for example A:1), otherwise DOSplus will treat the rootname as a drive number.

## 3.3 Listing Disk Contents

As information is stored on disk in files, there is a need to be able to easily find out what files are stored on a disk. The command

### DIR drive

where **drive** is the number of the disk drive containing the disk of interest, is used for this purpose. This command will output the disk number, name and a list of all the files on the disk in the specified drive. It will also output a count of the files on the disk and the number of bytes unused (this latter information is also obtainable by using the **FREE** function – see 3.4). The default value of **drive** is **DEFD** (see **DRIVE** in 2.2).

Each file in the directory listing has a number alongside. This is the file length, or the number of bytes used by that file (also obtainable through the **LOF** function – see 3.5). The number of bytes allocated to the file is usually larger and is always a multiple of complete sectors. If the directory contains a large number of files, screen output may be halted temporarily by using <SHIFT> and <@> (press any key to restart) or terminated by using <BREAK>. Alternatively, use the '/' option to page the listing to the screen, for example

### DIR / 1

Again, press any key (except <BREAK>) to output the next page.

The DIR command may be directed to send its output to any open device or file, see the command specification for full details.

## 3.4 Disk Free Space

As mentioned above, as well as through the **DIR** command, the amount of free space on a disk may be found using the function **FREE**. For example,

### PRINT FREE drive



will give the number of bytes unused on the disk in drive **drive** (the default for **drive** is **DEFD**). This function may be used to advantage in a program to check the free space before committing data to a new file. For example,

```
100 IF FREE 2 < 5000 THEN PRINT "NOT ENOUGH DISK SPACE"
```

### 3.5 File Lengths

As well as through **DIR**, it is also possible to obtain the length of an individual file through the function **LOF**. For example

```
PRINT LOF "DATABASE.DAT"
```

will give the length of file **DATABASE.DAT**. The extension (for example **DAT**) must be specified as there is no default with **LOF**. This function may also be used in programs, for example

```
100 F$ = "PROGRAM.BAS"  
110 PRINT CHR$(34); F$ ;CHR$(34);" OCCUPIES"; LOF(F$) ;"BYTES"
```

Note that parentheses are optional with **LOF**.

### 3.6 Copying Files

To transfer a file from one disk to another, or to duplicate a file giving it a different name, the **COPY** command is used. The command

```
COPY "OLDPROG.BAS" TO "NEWPROG.BAS"
```

will create the file **NEWPROG.BAS** as a copy of **OLDPROG.BAS** on the same disk. Note that the extension (in this case **BAS**) of both files must be specified, as there is no default with the **COPY** command (or, more strictly, the default is null). Also, the new extension need not be the same as the old one, though it is usual for this to be the case to avoid confusion.

The copy command can be used to transfer files from one disk to another. For example, when more than one disk drive is available,

```
COPY "3:PROGRAM.BAS" TO "2:PROGRAM.BAS"
```

will copy **PROGRAM.BAS** from the disk in drive 3 to the disk in drive 2, while to copy from one disk to another using only one drive, use

```
COPY / "PROGRAM.BAS" TO "PROGRAM.BAS"
```

### 3.7 Changing a File Name

If you wish to change name of a file, this can be done by copying the file to a file with the new name and deleting the original file, or more quickly by using the command **RENAME**:

```
RENAME "PROGRAM.BAS" TO "SPOOL.UTY"
```

This replaces the name **PROGRAM.BAS** by **SPOOL.UTY** in the directory entry for the file. This removes the need to perform the file to file copy, followed by the delete action.

**RENAME** is also used to rename disks (see 6.11).

### 3.8 Deleting Files

If you create a large number of files on a disk, the disk or the directory may fill up. The state of the disk can be monitored by using **FREE** or **DIR** to check on the free space and **DIR** to monitor the number of files. If you try to save a file which is larger than the available space, the file will be created, taking up all the space on the disk, and a **DF ERROR** will occur. The file created will not be complete and should be deleted and re-saved on another disk, or the same disk if other files are also deleted to make enough space available. If the directory becomes full, a **FD ERROR** will occur. The directory will hold up to 160 entries (80 for single density), whatever the disk format, but files may use more than one entry when the disk space becomes fragmented as files are created and deleted.

To alleviate the situation, do not keep files unnecessarily. For example, when you are sure that **BAK** files are not needed, delete them. This is done with the **KILL** command. As an example,

**KILL "PROGRAM.BAK"**

will delete the file **PROGRAM.BAK**. As there is no default extension for the **KILL** command, the extension (**BAK** in this example) must be included. If you have a large number of files to **KILL**, then **KILL /** or **KILL / drive** can be used. This will work through the directory of the disk in the specified drive, listing the files and waiting for a key press. If you press **<Y>**, the file will be **KILLED**, any other key (except **<BREAK>**) will move to the next file and **<BREAK>** will terminate the command immediately. Files which are write or delete protected (see below) are automatically passed by the command.

### 3.9 Delete Protecting Files

The **PROTECT** command provides a facility to avoid losing many hours of work accidentally, for example typing **KILL "PROGRAM.BAS"** instead of **KILL "PROGRAM.BAK"** or **'Y'** instead of **'N'** in response to a **'KILL /'** prompt. Any file may be protected from deletion (or over-writing) by using the command

**PROTECT "MONEY.BAG" or PROTECT ON "MONEY.BAG"**

(**ON** is the default). Commands such as **KILL "MONEY.BAG"** will now generate an error, until the protection is removed by

**PROTECT OFF "MONEY.BAG"**

While a file is protected, a lower case **'p'** (an inverse video **'P'** on the normal **DRAGON** screen) will appear alongside the filename in the directory listing.

### 4.0 Program Files

There are five disk commands associated with saving, loading and running programs. The three commands **SAVE**, **LOAD** and **RUN** may be used with both **BASIC** and machine code programs, while **CHAIN** and **MERGE** are limited to **BASIC** programs. Files written to disk by the **SAVE** command have a nine byte header, the format of which is defined by the syntax of the **SAVE** command, and which is used by **LOAD** to distinguish between **BASIC** and machine code programs (details of this header are given in Appendix 4).

#### 4.1 Storing a BASIC Program on Disk

A **BASIC** program may be saved on disk by using the command **SAVE**. For example

**SAVE "BASPROG"**

will write the **BASIC** program in memory to disk, assigning it the filename **'BASPROG.BAS'**.



The name 'BASPROG' can be any valid name, as defined in 3.2 above. After you press <ENTER> there will be a short delay, while the program is written to disk, and **OK** will appear on the display. Unlike cassette, even large programs will take only few seconds to be written to disk. If you now use the DIR command to list the files on the disk, the file BASPROG.BAS will be included in the list of names displayed. The extension BAS has been allocated by default. You do not have to adhere to the default extensions, but can use your own to over-ride the defaults. For example

```
SAVE "PROGRAM.COMD" or SAVE "PROGRAM.UTY"
```

It is also possible to store BASIC programs as text files. For example,

```
OPEN "O",#3,"BASPROG.TXT" : POKE 111,3 : LIST
```

## 4.2 Loading and Running a BASIC Program

A BASIC program which has been saved on disk may be loaded into memory and run by using the command sequence

```
LOAD "GRAPHS"  
RUN line number
```

where 'GRAPHS.BAS' is the file containing the BASIC program and **line number** is the line at which the program is to start. However, if you wish to load and immediately run a BASIC program, you can use a shorthand form where the RUN command both loads and enters the program. For example, the command

```
RUN "GRAPHS",line number
```

will perform the same action as the LOAD and RUN sequence above. If the program is to run from the first line, then the line number and the preceding comma may be omitted.

To load a BASIC program stored as a text file, the sequence

```
OPEN "I",#5,"BASPROG.TXT" : POKE 111,5 : EXEC 33658
```

is used. this will merge the file program as if it had been typed at the keyboard. Note that both save and load text file options may also be used with cassette.

## 4.3 Calling Programs in Sequence

The **CHAIN** command is used for running programs while retaining the values of any variables already set up. This command is frequently used for suites of programs which run in sequence. Note also that data files are left open when programs are CHAINED. The command

```
CHAIN "GRAPHS"
```

will load and run the BASIC program in the file **GRAPHS.BAS**, retaining any variable values already set up. As an example of the use of CHAIN, try the following

```
100 INPUT "X";X  
110 INPUT "Y";Y  
120 INPUT "Z";Z
```

```
SAVE "INPUT"
```

```
130 IF X>Y THEN IF X>Z THEN M=X ELSE M=Z  
    ELSE IF Y>Z THEN M=Y ELSE M=Z  
140 PRINT "MAX VALUE =";M
```

```
SAVE "MAXIMUM"
```

```

130 IF X<Y THEN IF X<Z THEN M=X ELSE M=Z
      ELSE IF Y<Z THEN M=Y ELSE M=Z
140 PRINT "MIN VALUE =" ;M

```

```
SAVE "MINIMUM"
```

You now have three files: INPUT.BAS, MAXIMUM.BAS and MINIMUM.BAS.

You can input values for X, Y and Z by using

```

RUN "INPUT"
X?21<ENTER>
Y?7<ENTER>
Z?14<ENTER>

```

As the variables are not cleared when a program finishes, the CHAIN command can now be used to work out the maximum or minimum values. For example, the command

```
CHAIN "MAXIMUM"
```

will give

```
MAX VALUE = 21
```

and the command

```
CHAIN "MINIMUM"
```

will give

```
MIN VALUE = 7
```

For this example, the technique is probably over complicated, but for programs involving large quantities of data it can be very useful, especially if the total space needed by a single program and its associated data would be larger than the available memory.

Note that it is also possible to use CHAIN with a second parameter:

```
CHAIN "PROGRAM",entry line
```

where **entry** is the line number at which the program is to start running.

## 4.4 Combining BASIC Programs

Two BASIC programs may be combined by using the **MERGE** command. This will load the BASIC program from the file specified in the command and superimpose this onto the program already in memory, as if the program taken from the disk file had been typed at the keyboard. This means that if each program contains a line with the same line number, the line already in memory will be replaced by the one from the disk file.

For example, using the programs from CHAIN,

```

LOAD "INPUT"
MERGE "MINIMUM"

```

will be the following program in memory:

```

100 INPUT "X";X
110 INPUT "Y";Y
120 INPUT "Z";Z
130 IF X<Y THEN IF X<Z THEN M=X ELSE M=Z
      ELSE IF Y<Z THEN M=Y ELSE M=Z
140 PRINT "MIN VALUE =" ;M

```



If the command

```
MERGE "MAXIMUM"
```

is now used, the program will be

```
100 INPUT "X";X
110 INPUT "Y";Y
120 INPUT "Z";Z
130 IF X>Y THEN IF X>Z THEN M=X ELSE M=Z
      ELSE IF Y>Z THEN M=Y ELSE M=Z
140 PRINT "MAX VALUE =";M
```

## 4.5 Binary Files

Machine code programs, graphics screens or any area of memory may be saved on disk using the

```
SAVE "SCREEN",start,end,entry
```

form of the SAVE command. Here, **start** is the starting address of the code in memory, **end** is the address of the byte following the last byte of memory to be written to disk and **entry** is the address at which the program should be entered when run (the default value used by EXEC). 'SCREEN' may, of course, be any valid filename.

With this form of the SAVE command, the file will be given the default extension BIN (a binary file). Binary files may be loaded using

```
LOAD "SCREEN.BIN",begin
or RUN "SCREEN.BIN",begin
```

where **begin** is an optional load address. Note that the **BIN** extension must be supplied as **LOAD** uses **BAS** by default. The default value for **begin** is the address of the start of the original code (the value of **start** when it was saved). The default EXEC address is calculated by adding the offset '**begin** - **start**' to the **entry** address in the file.

Two points to bear in mind if you are used to **CSAVEM** and **CLOADM** are:

- 1) the **end** address with **CSAVEM** is '**start** + **length** - 1' (ie one less than with **SAVE**)
- and 2) the **begin** address with **LOAD** is **not** an offset as it is with **CLOADM**.

## 4.6 File Backup

If you **SAVE** a BASIC or machine code program and specify a filename which is already in use on the disk, the existing file is renamed and given the extension **BAK**, and a new file is written with a **BAS** or **BIN** extension. Any existing **BAK** file with the same rootname has its directory entry deleted. For example, suppose **FINANCE.BAS** already exists, and you save a new program using

```
SAVE "FINANCE"
```

There will now be two files in the directory of the disk: **FINANCE.BAK** and **FINANCE.BAS**, where **FINANCE.BAS** is the program just saved and **FINANCE.BAK** is the previous **BAS** version.

If, at some later time, you use an identical command to save another program there will still be two versions on the disk, but now **FINANCE.BAS** will be the last version saved and the previous **BAS** version will be **FINANCE.BAK**. Details of the 'two versions ago' **BAK** file will have been deleted from the directory on the disk.

The backup version may be used, in preference to the latest version, by including the **BAK** extension when referencing the file, for example

```
LOAD "FINANCE.BAK"
```

If you need to go back to your previous program because you have corrupted your latest version, it is always available (and can be re-saved as the latest). Obviously, you can use RUN, CHAIN or MERGE with a backup file, by using the BAK extension, in a similar way.

An important point to note is that the backup system renames files to a BAK extension for BASIC, binary and data file backup. This means that if two or more files of different extensions have the same rootname, the backup files will have identical file names and cannot co-exist; that is, only the last file created will have a backup version.

## 5.0 Data Files

The combination of a directory of files and the ability to quickly access any sector of the disk (and replace the data 'in situ') is not only beneficial for program storage and running, but conveys a number of advantages for data files. The two most important are the reduction in data access times and the ability to have more than one file open at a time – up to **ten** files may be in use simultaneously.

### 5.1 Opening Files

Before a file can be used for data transfer, it has to be opened. This may be done implicitly through **FREAD** or **FWRITE** (see 5.4) or explicitly by an **OPEN** command (as with cassette files). For example

**OPEN mode,stream,filename**

where **mode** defines which of the four modes is used to open the file (see OPEN command details), **stream** is the file control block (in the range 1 to 10) to be used and is the number used by INPUT, PRINT etc when accessing the file and **filename** is the file to be used (default extension is DAT). Each time a file is opened, one of the ten 'file control blocks' is allocated to the file to hold information needed by DOSplus, for example – the read pointer (see 5.6). A file opened explicitly can also be accessed by name, using **FREAD**, **FWRITE** etc.

Note that whatever mode is used to open a file, it is **always** available for reading and writing (subject to the normal protection status).

### 5.2 Creating Files

If a file does not exist when it is opened, it will be created as an empty file (unless opened for reading). However, when using the 'random access' facilities of **FREAD** and **FWRITE** (see 5.7), the file must be large enough to hold the data. For this situation, the **CREATE** command is used to create (and open) a file of the required size. For example

**CREATE "FILE",size**

creates a file FILE.DAT of length **size** (default 0).

### 5.3 Closing Files

As file control blocks remain open and only ten files are permitted at one time, there would seem to be a problem if more than ten files are needed for an application. This may be overcome by closing files, as they are finished with or temporarily, and then opening others that are needed. There are a number of ways of calling the **CLOSE** command – to close all files, to close only files on one particular disk, or to close individual files. For example,

**CLOSE**



will close all disk and cassette files currently open and set the status of all drives to 'unaccessed', while

#### **CLOSE drive**

closes all files on the disk in drive number **drive** (value 1 to 4) and sets its status to 'unaccessed'. In this case, cassette files or files on other disks are not affected. To close individual files, it is necessary to alter **CLFLG** (location 247 or \$F7) by a POKE (see the specification for the CLOSE command for details), after which, the command

#### **CLOSE stream,stream,stream**

where **stream** is value from 1 to 10 (or -1 for cassette files), will close the files associated with the specified streams (or file control blocks).

## **5.4 File Input and Output**

Data files can be read and written by both extended INPUT and PRINT commands and by the additional FREAD and FWRITE disk commands. There are subtle differences in the operation of these commands and you are directed to the command descriptions for this level of detail. One major difference has been mentioned above - FREAD and FWRITE will implicitly open a file to be used while INPUT and PRINT expect the file to have been opened by an explicit OPEN command (as with cassette files).

The program

```
100 FWRITE "FILE";A,B,C or 100 OPEN "A",#1,"FILE" : PRINT #1,A,B,C
```

will write the values of A, B and C to the end of the file FILE.DAT, creating a file of zero length if the file does not exist. Assuming this latter case, the program

```
100 FREAD "FILE";A or 100 OPEN "I",#1,"FILE" : INPUT #1,A
```

will read the three values from the file 'FILE.DAT' as a single number, because the FWRITE and PRINT commands do not normally write a terminator after each value, so that the 'random access' form of the command (see 5.7) can be used. To overcome this problem, a terminator must be written between the values output, either by setting '**cassette mode**' (see 5.8) or explicitly outputting the terminators as in the following programs

```
110 FWRITE "FILE";A,"","B","","C  
120 FREAD "FILE";P,Q,R
```

or

```
110 OPEN "A",#1,"FILE" : PRINT #1,"","A","","B","","C  
120 INPUT #1,P,Q,R
```

When using INPUT or FREAD, there is also a problem when a string contains a field terminating character (comma or colon). For example, running the following program

```
10 $$="THERE IS A COMMA (,) AND A COLON (:) IN THIS STRING"  
20 FWRITE "STRING";$$  
30 FREAD "STRING";R$  
40 PRINT R$
```

gives

```
THERE IS A COMMA (
```

as the comma has terminated the string input. The remainder of the output string is then treated as two further fields containing

```
and ) AND A COLON (  
 ) IN THIS STRING
```

There are two ways to overcome this problem, depending on whether the string is a complete record or just one field of a record. If the string is the complete record, then either

```
30 FLREAD "STRING";R$      or      20 FWRITE "STRING";CHR$(34),$$,CHR$(34)
```

may be used and either of these programs will give

```
THERE IS A COMMA (,) AND A COLON (:) IN THIS STRING
```

If the string is only one field of the record, then FLREAD is not suitable (unless the string is of fixed size – see 5.7) and the CHR\$(34) technique should be used to write a **'quoted string'**.

## 5.5 File Read Pointer Manipulation

The function LOC may be used to determine the position of the read pointer in a file, or to set the position. For example,

```
PRINT LOC "INFILE"
```

gives the position of the next byte to be read in INFILE.DAT, while

```
LOC("INFILE") = 0
```

will reset the pointer to the start of INFILE.DAT. This can be used where many files are required (more than ten) and a file needs to be closed temporarily. The statements

```
100 P=LOC(5) : CLOSE #5  
200 OPEN "I",#5,"INFILE" : LOC(5)=P
```

could be used, where stream 5 has been allocated to another file for processing between these statements.

## 5.6 'End of File' Processing

Often, the length of a file is not known, so there needs to be some means of determining when all the data has been read. This can be done by using LOF and LOC; they will be equal when the data is exhausted. However, as with cassette files, the EOF function may be used. In the statement

```
E = EOF("FILE")
```

E will usually take the value 0 ('false'), but if the read pointer is at the end of FILE.DAT (that is, there are no more records to read) it will have the value -1 ('true'). To use EOF, the file must already be open.

## 5.7 FREAD and FWRITE extensions

The commands FLREAD, FREAD and FWRITE may be used with parameters to specify a start position anywhere within the file, and the length of the record to process. These take the form

```
FLREAD "FILENAME",FROM start,FOR length:string  
FREAD  "FILENAME",FROM start,FOR length:variables  
FWRITE "FILENAME",FROM start,FOR length:variables
```

where **start** is the first byte in the file where the information is to be read from or written to, and **length** is the number of bytes to be added to the read pointer after a read operation, or



the number of bytes to be written to the file. Note that when **FOR** is used with a read operation, input will stop when either the variable list has been satisfied or the required number of characters has been input. Similarly, output will stop when the required number of characters has been output, including any <SPACE> characters written after the data to make the required length.

The following 'database' construction provides a simple illustration.

Two small programs are required. The first is used to create the data file and store data in it. The second is used to retrieve the data.

```
10 INPUT "FILENAME";N$
20 INPUT "NUMBER OF RECORDS";N
30 INPUT "RECORD LENGTH";L
40 CREATE N$,N*L+20
50 FWRITE N$,FROM 0;N : FWRITE N$,FROM 10;L
60 CLOSE N$ : CHAIN "UPDATE",30
```

SAVE "CREATE"

```
10 INPUT "FILENAME";N$
20 FREAD N$:N : FREAD N$,FROM 10;L
30 INPUT "RECORD NUMBER";I : IF I<1 THEN END
40 IF I>N THEN PRINT "RECORD NUMBER TOO BIG" : GOTO 30
50 PRINT "RECORD";I; : INPUT R$
60 FWRITE N$,FROM (I-1)*L+20,FOR L;R$
70 GOTO 30
```

SAVE "UPDATE"

```
10 INPUT "FILENAME";N$
20 FREAD N$,FROM 0;N : FREAD N$,FROM 10;L
30 INPUT "RECORD NUMBER";I : IF I<1 THEN END
40 IF I>N THEN PRINT "RECORD NUMBER TOO BIG" : GOTO 30
50 FLREAD N$,FROM (I-1)*L+20,FOR L;R$
60 PRINT R$ : GOTO 30
```

SAVE "READ"

The data file is created using

RUN "CREATE"

The program first asks for a filename (any legal filename may be given), the number of records and the record length. After creating the file and writing the number of records and record size, the update program is chained from line 50 to set up the initial data. To update records at a later time, UPDATE is run on its own. To test the program, enter 6 and 25 for the number and length of records, and enter the following records (in any order):

```
1 LONDON
2 MANCHESTER
3 BIRMINGHAM
4 EDINBURGH
5 CARDIFF
6 GLASGOW
```

Do not worry if you make mistakes, as records may be easily overwritten. For example, we could replace '3 BIRMINGHAM' with '3 SWANSEA' during the file creation (or a later update run). The program is terminated by entering a 0 for the record number. Note that the data does not need to be the length specified during the create phase, nor does a replacement item need to be the same length as the item replaced. The 'READ' program can now be run to display the records. The filename must be specified again, as 'RUN' deletes all variables, but the other parameters are read from the file. If a number from 1 to 6 is entered, the relevant record is listed, while 0 will terminate the program.

## 5.8 Print #n Modes

To ease the change from cassette to disk based systems for data handling, DOSplus includes a capability to treat disk files in the same way as cassette files. In addition to the availability of INPUT, PRINT etc., DOSplus also uses a 'flag' (**CMFLG** - location 2035 or \$7F3) to define whether 'end of line characters' are to be inserted between all variables output. To enable or disable the mode,

**POKE 2035,stream**

where a value for **stream** between 1 and 10 will enable the mode for only that stream; a value of 11 will enable the mode for all **FWRITE** operations and any other value will disable the mode. The default at power on or any **RESET** is 0 (mode disabled).

## 5.9 Machine Code Character I/O

Single character disk file I/O may be performed from machine code programs by using the standard character handling routines of BASIC. This is done by setting location 111 (\$6F) to the file control block number and using the calls: JSR \$B50A for input and JSR [\$A002] for output. The file control block number is returned in 'A' when JSR [\$C00A] is used to open a file control block. (See Appendix 2 for details of machine code facilities).

## 6.0 Miscellaneous Features

DOSplus contains a number of additional BASIC commands, functions and extensions to aid programming, described in this section.

### 6.1 Selective RESTORE

One of the enhancements provided by DOSplus is to the **RESTORE** command. It is now possible to move to the required position in your **DATA** statements without having to READ and discard preceding data. This is done by

**RESTORE line number**

where **line number** is the line at which READ processing should start. Note that the line specified must exist, but does not need to have a **DATA** statement on it. Line numbers used in **RESTORE** are renumbered by **RENUM** as usual.

### 6.2 Text Command Files

It is possible with DOSplus to read commands from a disk file instead of the keyboard. This is accomplished with the **FROM** command. For example,

**FROM "COMLIST"**

will take a line of input from the file **COMLIST.CMD** instead of the keyboard (**.CMD** is the default extension). This is not as restrictive as it may seem, as the line may contain a further **FROM** statement. The line to be obeyed must be 255 characters long (including any spaces used as padding) and must not contain any 'end of line' characters. A simple method to set up a file is to use a program such as

```
10 F$ = "STARTUP.CMD" : LINE INPUT CMD$ : FWRITE F$, FOR 255; CMD$;
```

Note the use of **FOR** to get 255 characters, with padding spaces, and the final ';' to suppress the 'end of line' character.



## 6.3 Write Verification

In the default condition, all BASIC commands which write to disk perform a 'read-after-write' verification. This does not compare the data read back from the disk with the data written, but verifies that no errors occurred during the recording process which will prevent the disk being read at a later date. It is very unlikely that the data recorded will be in error if the verification read is successful. This verification can be suppressed by using the command

### **VERIFY OFF**

and re-activated with

### **VERIFY or VERIFY ON**

Note that **directory tracks 16 and 20** (both for directory records and data records) are always verified, irrespective of the selected VERIFY status.

## 6.4 Disk Backup

To safeguard against that unfortunate accident, like your son using a disk as a 'Frisbee' or even answering 'Y' to the DSKINIT check when you meant 'N', it is best to keep duplicate copies of your important disks or files. This can be done by SAVEing programs onto separate disks or copying from one disk to another, or you can use the BACKUP command to 'duplicate' a complete disk. Note that BACKUP always verifies all tracks, irrespective of the VERIFY status. The command

### **BACKUP source TO dest,sides,tracks**

will create an exact copy of the disk in the **source** drive on the disk in the **dest** drive. The parameter **sides** is the number of sides (1 or 2, default 1) and the parameter **tracks** is the number of tracks (usually 40 or 80, default 40). The other default values are **source=DEFD** and **dest=source**. BACKUP will request confirmation with the message 'SURE?' before starting to copy data. Note that the sides and tracks parameters apply to both disks and that the destination disk must have been initialised to a suitable format by DSKINIT before BACKUP is invoked. If a disk with other than 40 or 80 tracks is to be duplicated, DOSplus will check by repeating the number of tracks and waiting for confirmation.

When **source** and **dest** are the same, some mechanism for using two disks in only one drive must be invoked. When the BACKUP command is used under these conditions, the DRAGON will prompt you to 'INSERT SOURCE' and 'INSERT DESTINATION' until the disk has been duplicated.

As the BACKUP command uses the space between the end of variables and the 'stack', to obtain faster copying delete the contents of memory by typing **NEW** and reduce string storage and graphics memory by **CLEAR 10,32767** and **PCLEAR 1** before starting BACKUP. For a single drive BACKUP, this will reduce the number of disk swaps to a minimum.

## 6.5 Pausing a Program

If your latest program is running too fast, you can slow it down by using the **WAIT** command. This pauses the program for a specified number of milliseconds. For example **WAIT 2000** halts the program for two seconds. This can be effective for delaying messages long enough for **<SHIFT> + <@>** to be used to hold the program indefinitely. For example

```
10 CLS
20 PRINT @128, "THIS MESSAGE WILL STAY ON", "SCREEN FOR 5 SECONDS"
30 WAIT 5000
40 CLS
```

## 6.6 Memory Allocation

Two additions which allow memory allocation to be found are the **HIMEM** and **FRES** functions.

The function **HIMEM** returns the highest memory location available to BASIC. On power up this has a value **32766**. If **CLEAR** is used with two parameters to reserve space for machine code or data, then **HIMEM** will return a value one less than the second parameter. For example the program

```
10 CLEAR 500,18000
20 PRINT "TOP OF MEMORY FOR BASIC =" ; HIMEM
```

gives the result

```
TOP OF MEMORY FOR BASIC = 17999
```

The second function, **FRES**, returns the number of bytes available for additional strings in the string space. Note that **FRES** is a numeric variable, not a string.

In order to calculate the free space, **FRES** forces a 'garbage collection' routine to run. This 'shuffles' all strings to the top of the string space, leaving a contiguous free area. Calling **FRES** can be used to reduce the noticeable effects of string tidying in programs with lots of string manipulation.

## 6.7 Error Trapping

Normally, when an error occurs, the program will stop and an error message will be output. If the error is caused by a program fault, then, obviously, the program needs to be corrected and re-run. However, if the error is due, for example, to incorrect data being typed in, then it would be useful to be able to continue, after taking appropriate action. For example, the program

```
10 INPUT "X";X
20 PRINT "THE SQUARE ROOT OF";X;"IS";SQR(X)
30 GOTO 10
```

would stop with a **?FC ERROR IN 20** if a negative value was entered. This could be tested each time through the loop with an **IF** statement, but that would slow the program (not much effect here, but could be noticeable in a large program). To detect errors efficiently, the statement **ERROR GOTO n** is used. This tells the **DRAGON** that if an error is detected control should be passed to line number **n**. Once this has occurred, there are two functions which help to identify the error: **ERL** gives the line number where the error occurred, and **ERR** gives the error code number (each type of error has a different code number – see Appendix 5 for a list of numbers).

The code for **FC ERROR** is 8 so the following lines may be added to the program to deal with the error.

```
5 ERROR GOTO 50
50 IF ERR=8 THEN PRINT "CANNOT CALCULATE IMAGINARY NUMBERS":GOTO 5
60 PRINT "ERROR NUMBER";ERR;"IN LINE";ERL
```

Note that error trapping is turned off by 'RUN' and as soon as the error trap routine is entered.

## 6.8 Warning 'Bell'

Where an audible warning is required, this can be done by using **SOUND** or more conveniently by the **BEEP** command. By default, a single 'beep' occurs or a number may be specified. For example, to produce **n** 'beeps', use the command

```
BEEP n
```



## 6.9 Data Exchange

There are many occasions, such as sorting data, when the values of two variables have to be exchanged. Without DOSplus, a temporary variable has to be used to hold one value while the other is transferred. With DOSplus, numeric and string variables or array elements (not complete arrays!) can be easily exchanged by using the command **SWAP**. For example

```
SWAP XS(1),XS(14)
```

will exchange the strings associated with the array elements **XS(1)** and **XS(14)**.

## 6.10 Automatic Line Numbering

When typing BASIC programs, a problem that occurs all too frequently is mistyping a line number. Where this results in 50 instead of 500, for example, not only does the line get placed in the wrong part of the program, it usually overwrites a long, complex statement (cf Murphy's Law!). **AUTO**, for automatic line numbering, is designed to eliminate this problem. The command

```
AUTO start,increment
```

will give you numbers starting with the line numbered **start** and incrementing by the amount **increment**. You just type your lines in, ending each line with **<ENTER>**. To exit from **AUTO**, just type **<ENTER>** immediately after the line number has appeared. Input when in **AUTO** mode is just like any keyboard input, and will insert or over-write lines as appropriate. However, if the line already exists, a '?' will be output before the line number. The default values for **start** and **increment** are 100 and 10 respectively.

The following example shows how **AUTO** can be used.

```
AUTO 100,10  
OK  
100 X2 = X*X  
110 X3 = X*X*X  
120 X4 = X*X*X*X  
130 <ENTER>
```

We may decide that we should have used subroutines for each power, so

```
AUTO 105,10  
OK  
105 RETURN  
115 RETURN  
125 RETURN  
135 <ENTER>
```

and, finally, some comments to identify what is being calculated:

```
AUTO 99,10  
OK  
99 REM X SQUARED  
109 REM X CUBED  
119 REM X TO FOURTH POWER  
129 <ENTER>
```

This gives the required program.

## 6.11 Naming and Numbering Disks

As an aid to remembering the general content of a disk it is useful to have a diskname in the directory listing. This is accomplished by means of the RENAME command, and

**RENAME #2 TO "ASSEMBLR.MAS"**

will write the name 'ASSEMBLR.MAS' to the directory track of the disk in drive 2, without affecting the disk number or the remainder of the directory contents.

While naming disks is an aid to remembering the disks contents, it is generally more convenient to use a number to locate a disk in a library box. To this end, DOSplus uses a previously unused byte in the first directory sector to hold an 8 bit positive disk number (range 0 to 255), which is output in the directory header by DIR. This can be set by RENAME at the same time as the diskname is written (but cannot be set independently). The command

**RENAME #3 TO "GAMES.OLD",53**

will rename the disk in drive 3 to 'GAMES.OLD' and write 53 as its number.

## 6.12 Booting an Operating System

It is possible to run the DRAGON with a different operating system to the BASIC and DOSplus combination initialised at power up, and DOSplus has a BOOT command for this purpose. The command

**BOOT drive**

where **drive** is (optionally) the number of a disk drive, loads and runs a 'bootstrap' routine from the disk in the specified drive (default is DEFDD). DEFDD is set to **drive** by BOOT before the bootstrap routine is loaded. This means that if the BOOT fails for any reason, the default drive will now be set to the drive specified in the command.

It is also possible to run any BASIC or machine code program on a disk by setting up a suitable bootstrap routine. For example, the program

```
10 CLEAR 500 : INPUT "DISK"; D
20 SREAD D,20,1,A$,B$ : IF ASC(LEFT$(A$,1)) AND 4 THEN 40
30 PRINT "BOOTSTRAP SECTOR IN USE" : END
40 MID$(A$,1,1) = CHR$(ASC(LEFT$(A$,1)) AND 251)
50 SWRITE D,20,1,A$,B$ : A$ = "OS"
60 FOR I = 1 TO 12 : READ A : A$ = A$ + CHR$(A) : NEXT I
70 A$ = LEFT$(A$ + STRING$(128,0),124) + "RUN" + CHR$(34)
80 INPUT "FILE";F$ : SWRITE D,0,3,A$,F$ : END
90 DATA 220,138,221,111,28,254,142,38,123,126,131,125
```

will write a bootstrap to run a program in the specified file. The name of the file containing the program to be run may be easily changed by using

```
10 CLEAR 500 : INPUT "FILE";F$ : INPUT "DISK";D
20 SREAD D,0,3,A$,B$ : SWRITE D,0,3,A$,F$
```

Note that BOOT is automatically invoked at power on or any cold reset.

DOSplus DELTA users should note that due to the different addressing and limitations in the DELTA cartridge, it is not possible to run FLEX or OS9 in their standard form, nor can BASIC 42 be used with double density disks.



## Appendix 1

### DOSplus Extensions to BASIC

This Appendix lists alphabetically the commands and functions provided by DOSplus, through which BASIC accesses disks, with a definition and description of each. Within the formal definition, square brackets [ ] are used to denote optional parameters. Note that **DEFD** is used throughout this Appendix to indicate the default drive number, as set by the **DRIVE** command.

---

## AUTO

Format: **AUTO** [start [,increment]]

Defaults: start = 100  
increment = 10

The **AUTO** command automatically generates line numbers for a program being input from the keyboard. The first line will have the line number '**start**' and the interval between line numbers will be '**increment**'. Any lines in the program with corresponding line numbers will be replaced by the input line in the normal way. **AUTO** is terminated by pressing <ENTER> or <BREAK> as the first character on the line, or by the line number exceeding 63999. Pressing <BREAK> or <SHIFT>+← after a character has been inserted on the line will move to the next line, WITHOUT DELETING ANY EXISTING LINE WITH THE SAME LINE NUMBER. All other line editing is permitted as normal. If a line already exists with the line number being generated, a '?' will be output before the line number.

Example:

**AUTO 20,15**

will produce successive line numbers 20,35,50 etc.

---

## BACKUP

Format: **BACKUP** [source [TO destination [,sides [,tracks]]]]

Defaults: source = **DEFD**  
destination = source  
sides = 1  
tracks = 40

**BACKUP** will perform a sector by sector copy of the source disk onto the destination disk. If the number of tracks is not 40 or 80, the number will be displayed - type 'Y' to continue or <BREAK> to abort. Before performing the copy, **BACKUP** will output '**SURE?**'. Again type 'Y' to continue or <BREAK> to abort. If only one drive is used, screen prompts are used to specify which disk is required to be loaded and in this case the backup may be aborted by pressing <BREAK>. The destination drive must be capable of writing a disk with the same format as the source disk. **BACKUP** ignores the verify flag and all sectors will be read-after-write verified.

Examples:

**BACKUP 2 TO 4,1,80**

will produce an exact copy of the single sided 80 track disk in drive 2 on the disk in drive 4.

**BACKUP**

will duplicate a 40 track single sided disk on another disk, using only the default drive and prompting when a disk change is needed.

## BACKUP DIR

Format: BACKUP DIR [-] [drive]

Default: drive = DEFD

BACKUP DIR is used to backup or restore the directory track on a disk with duplicate directory tracks. The optional '-' parameter causes a restore to take place by copying the directory on track 16 to track 20, otherwise a backup takes place. If duplicate directories are not supported by the disk, a **?DB ERROR** will be reported

Examples:

BACKUP DIR

will copy the directory on track 20 of the disk in the default drive (DEFD) onto track 16.

BACKUP DIR - 2

will restore the directory on the disk in drive 2.

---

## BEEP

Format: BEEP [number]

Default: number = 1

BEEP causes the specified number of 'beeps' (maximum 255) to be emitted by the DRAGON.

Example:

BEEP 10

will cause the DRAGON to sound 10 'beeps'

---

## BOOT

Format: BOOT [drive]

Default: drive = DEFD

BOOT loads the program contained in sectors 3 to 18 (3 to 10 single density) of track zero of the disk in the specified drive into memory from address 9728 on; sets DEFD to 'drive' and starts running the loaded program from memory address 9730. For the program to be loaded, sector 3 **must** contain the characters **OS** as the first two bytes.

Example:

BOOT 3

will load and run the program on track 0 of the disk in drive 3.

## CHAIN

Format: CHAIN filespec [,start]

Default: start = first program line

CHAIN is used to load and run a BASIC program from the specified disk file, preserving the variables (including strings). The chained program will start running from the line specified in the command or the first line if no start line is given.

Example:

```
CHAIN "PROGA",500
```

will load the BASIC program PROGA.BAS into memory and start it running from line 500.

---

## CLOSE

Format: CLOSE [stream [,stream [, ...] ]]

Default: all streams

The CLOSE command has three modes of operation. Two of these are for compatibility with DRAGONDOS 1.0 and 1.3 (or later), and one is special to DOSplus. In the power on mode, full DRAGONDOS 1.0 compatibility is maintained, where the stream numbers 1 to 4 refer to disk drives, and all files on the disk in the specified drive are closed with the drive status being set to 'unaccessed'. Altering the value of memory location 247 (\$F7) to a value from 1 to 127 will enable the 'last file' mode, where the last disk file accessed is closed irrespective of the stream number. Finally, if location 247 is set to a value from 128 to 255, the file allocated to the specified disk control block is closed (see also OPEN).

Example:

```
POKE 247,128 : CLOSE 3
```

will close the file opened as stream #3

---

## COPY

Format: COPY [/] source file TO destination file

COPY allows files to be copied from one disk to another or duplicate files to be created on a single disk. There are no default file extensions for COPY, so the full filename is required. The optional '/' parameter will copy a file onto a second disk using a single drive, issuing prompts for a disk change as required. To ensure file integrity, COPY closes all files.

Example:

```
COPY "1:FILE1.BIN" TO "2:FILE7.BIN"
```

will copy the file FILE1.BIN from the disk in drive 1 into a file called FILE7.BIN on the disk in drive 2.



## CREATE

Format: CREATE filespec [,length]

Defaults: length = 0  
file extension = .DAT

CREATE is used to create a file of a specified size, usually for non-serial file operations. As CREATE will rename an existing file with the same file specification to type .BAK, it can also be used to 'empty' a file while retaining the previous version in one step. CREATE checks for sufficient space on the disk before taking any other action.

Example:

```
CREATE "4:LONGFILE",200000
```

will create a file LONGFILE.DAT of length 200000 bytes on a disk in drive 4 (provided there is sufficient space available)

---

## DIR

Format: DIR [/][drive]  
DIR [/][#stream [,drive]]

Defaults: stream = 0  
drive = DEF0

DIR is used to list the files on a disk in the specified drive. The output gives the optional disk number and name, a list of files with their protection (a lower case or inverse P if write/delete protection is set) and size in bytes, followed by the number of files on the disk and the amount of free space left (in multiples of 256 bytes). If the output is directed to the screen then the display will scroll slowly to allow the details to be read as they scroll. If the optional '/' parameter is used to output to the screen, 12 line paging will be enabled (the first page will contain 10 files), requiring a key to be pressed to continue to the next page. Screen output may be aborted by pressing the <BREAK> key or held with <SHIFT> + <@>. For printer output, the '/' parameter causes the directory to be printed in a single column, instead of the standard three columns. Note that '/' is ignored if output is not to the screen or the printer.

A typical listing of a directory of files is:-

```
DISK: 147 GAMES .MAS
```

```
SNAILS .BAK 2015
ZAPPIT .BAK 2566
SNAILS .BAS p 2015
DATAFILE .DAT 297
ZAPPIT .BIN 2566
```

```
5 FILES, BYTES FREE 164864
```

Examples:

```
DIR #-2,3
```

will list the directory of the disk in drive 3 to a printer, three filenames per line.

```
DIR /
```

will list the directory of the disk in the default drive to the screen, waiting for a key press after the first 10 files and then after every 12 files.

## DRIVE

Format: DRIVE drive

The DRIVE command is used to set the **default drive number** (mnemonic DEFD, location 1546 or \$60A) to be used with disk commands, in the range 1 to 4 (not 0 to 3 as used by other operating systems). At power on, the default drive number is set to 1.

Example:

DRIVE 2

will cause all disk accesses to default to drive number 2.

---

## DSKINIT

Format: DSKINIT [/] [drive [,sides [,tracks]]]

Defaults: drive = DEFD  
sides = 1  
tracks = 40

DSKINIT is used to format a disk for use with DOSplus, and set up an empty directory on track 20. Until formatted, a disk cannot be used as the necessary track and sector information is not available on the disk. If the number of tracks specified is not 40 or 80, confirmation that the number is correct will be requested, type 'Y' to accept the number or <BREAK> to abort without error. If the disk has previously been formatted, the message **SURE?** will be displayed. Type 'Y' to format the disk, <BREAK> to abort without error or any other key to abort with a **?PR ERROR**. While formatting takes place, the screen will display the track being written or checked.

DSKINIT assumes that a disk has not been previously formatted if an attempt to read sector 3 on track zero returns a **?RF ERROR**. In this case, there will be a delay of about seven seconds before formatting starts. If the delay occurs before the **SURE?** message, then an error other than **?RF ERROR** has occurred. This will not automatically abort the formatting, as this error may be the reason the disk is being re-formatted.

If the optional '/' parameter is used, track 16 will be allocated to data space adding 4½k (2½k single density) to the available data space.

Note that an empty directory is **not** written to track 16, so **BACKUP DIR must** be used before the directory recovery procedures become operative. Note also that DSKINIT does not check that the sides/tracks information matches the drive, but assumes that the information supplied is correct.

Examples:

DSKINIT 2,1,80

will format a single sided 80 track disk in drive 2. The first 18 sectors (10 single density) of track 16 are allocated for use as a duplicate directory, while

DSKINIT /

will format a 40 track single sided disk in the default drive, allocating track 16 for data storage.



## EOF

Format: E = EOF(filespec)  
E = EOF(stream)

EOF is a function which can be used to check the status of a file which is being read. EOF will return a value of 0 (false) if the file contains more data to be read, or -1 (true) if the end of the data has been reached. Note that the file must already have been opened before EOF is used, either by the use of OPEN or by FREAD, FLREAD or FWRITE.

Example:

```
IF EOF(6) THEN CLOSE #6
```

will close the file associated with stream 6 when the end of file is reached, provided memory location 247 contains a number between 128 and 255 (see CLOSE).

---

## ERL

Format: E = ERL

The function ERL can be used to get the number of the line of BASIC which caused an error to occur. It is normally used from within an error trap routine (see also ERR and ERROR GOTO).

Example:

```
PRINT ERL
```

will output the value 95 if an error occurred on line 95 of the program.

---

## ERR

Format: E = ERR

ERR is used to return the code number of an error when an error occurs. It is normally used within an error trap routine (see also ERL and ERROR GOTO).

The error codes are listed in Appendix 5.

Example:

```
PRINT ERR
```

will output a value of 148 if this statement is obeyed following a ?DF ERROR occurring.

## ERROR GOTO

Format:     **ERROR GOTO** line number

**ERROR GOTO** is used to point to an error trap routine, which is entered if an error occurs by obeying the last 'GOTO line number' specified in an **ERROR GOTO** statement. By using multiple **ERROR GOTO** statements, different trap routines may be used at selected points of a program. Error trapping is turned off by the **RUN** command, by specifying a line number of zero or when an error occurs (see also **ERL** and **ERR**). Note that **FOR** loop information is cleared by an error, resulting in **?NF ERROR** if an attempt is made to resume the loop through **ERROR GOTO**.

Example:

**ERROR GOTO 600**

will cause a **GOTO 600** to be obeyed if an error occurs.

---

## FLREAD

Format:     **FLREAD** filespec [,FROM start][,FOR length] ; string

Defaults:   start = read pointer  
             length = to next 'end of line'  
             file extension = .DAT

**FLREAD** reads a record from the file specified into a string variable. Input is terminated if an 'end of line' character is encountered or the **FOR** length expires (whichever comes first). The read pointer is advanced by the **FOR** length or to the character following the 'end of line' character if no **FOR** length is specified. The **FOR** length must be in the range 1 to 255.

Examples:

**FLREAD "FILE1",FROM 20,FOR 45;R\$**

will read the 45 bytes (or as many bytes occur before an 'end of line' character, if less) from byte 20 of the file 'FILE1.DAT' into the string **R\$**. The read pointer will be left at byte 65.

**FLREAD "FILE2.TXT";L\$**

will transfer data from the current read pointer position in file **FILE2.TXT** into the string **L\$** until an 'end of line' character is met.

## FREAD

Format: FREAD filespec [,FROM start][,FOR length] ; variables

Defaults: start = read pointer  
length = to next 'end of line' character  
file extension = .DAT

FREAD reads numeric or string fields from file records into the specified variables. At the end of the input, the read pointer is advanced by the FOR length (if specified) or to the character following the last field terminator. Fields are terminated by 'end of line' characters and by commas and colons if they do not occur between string quotes. If the FOR length exhausts before the variable list, the remaining variables in the list take a value of zero or a null string, as appropriate. The FOR length must be in the range 1 to 255.

Examples:

```
FREAD "TEXT.TXT";A,B
```

will read two fields from the current read pointer position in the file TEXT.TXT into the variables A and B.

```
FREAD "FILE",FOR 8,FROM 10;A,B$,C$
```

will read three fields from the file FILE.DAT into variables A, B\$ and C\$ respectively, leaving the read pointer at byte 18.

---

## FRES

Format: F = FRES

FRES is a NUMERIC function (despite the '\$') which is used to obtain the number of bytes of string space unused. In order to perform the calculation required, FRES packs the strings at the top of the string space by calling the 'garbage collection' routine. This function may also be used to force the tidying of the string space at times convenient to the user, rather than allow this to occur at random times.

Example:

```
X = FRES
```

will set X to the number of bytes available for strings.



## FREE

Format: F = FREE [drive]

Default: drive = DEFDD

FREE is used to obtain the number of unused bytes on a disk in the specified drive. Note that this value is always a multiple of 256 bytes (ie complete sectors) and does not include unused space in sectors at the end of files.

Example:

```
PRINT FREE 4
```

will print the number of bytes free on the disk in drive 4.

---

## FROM

Format: FROM filespec

Default: file extension = .CMD

FROM is used to obey a 255 character line from a command file in preference to a line of keyboard input. A single statement only may be obeyed, but this may include a further FROM call. Once loaded into the keyboard buffer, the normal keyboard input rules apply. Note that the line **must** not end with an 'end of line' terminator and should be padded to 255 characters with spaces (see FWRITE/FOR).

Example: \*

```
FROM "STARTUP"
```

---

## FWRITE

Format: FWRITE filespec [,FROM start][,FOR length];[USING format;][variable list]

Defaults: file extension = .DAT  
start = end of file  
length = as required

FWRITE is used to write data to a disk file and may be used in DIRECT mode. If the file does not exist, it is created with zero length, if it does exist, it is opened. No backup file is created. If FROM is specified, the file must be at least 'start - 1' bytes long. If FOR is specified, it must be in the range 1 to 255, and variables not output when the count expires will be ignored, or the record will be packed with spaces if the list expires first. If USING is included, the layout characters are exactly as used with PRINT. Note that FWRITE output to a disk file does not normally insert terminators between variables. If CMFLG (at 2035 or \$7F3) is set to 11 then 'end of line' characters will be inserted between variables automatically.

Example:

```
FWRITE "TEST";"TEST WRITE"
```

will write the string TEST WRITE and an 'end of line' character to the end of the file TEST.DAT

## HIMEM

Format: T = HIMEM

HIMEM is a function used to return the highest memory location available to BASIC. If CLEAR has been used with two parameters to alter the 'top of memory', this will return a value of one less than the second parameter value, otherwise a value of 32766 will be returned.

Example:

PRINT HIMEM

---

## INPUT

Format: INPUT #stream, variable list

Default: stream = 0

INPUT can be used to input data from a disk file in the same way as from a cassette file. A stream used to get data from a disk file in an INPUT statement must be associated with the disk file through the OPEN command. Data are separated by 'end of line' characters and, except when between string quotes, commas and colons. Note that, as with cassette and keyboard, INPUT reads at least one complete record (ie up to an 'end of line' character) from disk and discards any unused data after satisfying the variables list.

Example:

INPUT #4,NAME\$

will input the next field from the file associated with stream 4, storing it as a string in NAME\$

---

## KILL

Format: KILL filespec  
KILL / [drive]

KILL is used to delete unwanted files from a disk, releasing the space used by the file for re-allocation to a new file. KILL will operate in two modes - standard or 'delete with confirm'. If a standard file specification is supplied, the specified file is deleted (if not protected). If the '/' option is used, KILL will list each file on the disk in turn and, if the file is not protected, wait for a key to be pressed. At this point, pressing the <BREAK> key will abort the command, the <Y> key will KILL the file or any other key will move to the next file in the directory.

Example:

KILL "3:PROG1.BAK"

will delete the file PROG1.BAK from the disk in drive 3.

## LINE INPUT

Format: `LINE INPUT #stream, string variable`

Default: `stream = 0`

**LINE INPUT** is used to read a complete record from a disk file into a string variable. Input is terminated when an 'end of line' character is encountered. **LINE INPUT** may be used as an alternative to **FLREAD**, if **FOR** is not required. The stream number specified must be associated with a disk file by use of an **OPEN** command. Note that **FROM** may be emulated by using the **LOC** function to set the read pointer.

Example:

```
LINE INPUT #10,LIN$
```

will read from the current read pointer position to an 'end of line' character and store the data in the string **LIN\$**.

---

## LOAD

Format: `LOAD filespec [,binary load address]`

Defaults: `file extension = .BAS`

`binary load address = start address used when the program was SAVED`

The **LOAD** command is used to transfer the contents of a **BASIC** or **Binary** file into memory from disk. If the file specified contains a **BASIC** program, any existing **BASIC** program in memory will be deleted first (see also **CHAIN** and **MERGE**).

If the file contains a **Binary** program, this is loaded into memory and the **EXEC** address set to the value specified in the **SAVE** command. The optional **Binary** load address parameter may be used to over-ride the file load address, in which case the file entry address is adjusted by an equivalent offset. Note that with segmented binary files, any offset is calculated from the load address of the first segment and all segments are loaded using this calculated offset. The entry address is taken from the terminating null segment.

Examples:

```
LOAD "BASPROG"
```

will load the **BASIC** program **BASPROG.BAS** from the disk in the default drive

```
LOAD "4:MULTISEG.BIN",1536
```

will load the segmented binary file **MULTISEG.BIN** from the disk in drive 4, loading the first segment from address 1536 and applying the same offset to the other segments and the entry address.



## LOC

Format: L = LOC filespec                      LOC filespec = value  
         L = LOC (filespec)                  LOC (filespec) = value  
         L = LOC stream                      LOC stream = value  
         L = LOC (stream)                   LOC (stream) = value

LOC returns the value of the read pointer for a file or sets the read pointer to the specified value, either by file specification or stream number reference. The LOC parameter may optionally be enclosed in parentheses. Note that LOC will not open a file.

Example:

P = LOC 5                      or                      P = LOC(5)

will return the position of the read pointer for stream 5.

---

## LOF

Format: L = LOF filespec  
         L = LOF (filespec)

LOF is used to get the length of a file in bytes. If necessary, the file is opened to get the required information.

Example:

L = LOF "TELEWRIT.BIN"

will return the length of the file TELEWRIT.BIN to the variable L.

---

## MERGE

Format: MERGE filespec

Default: file extension = .BAS

MERGE is used to superimpose a BASIC program in the specified disk file on a BASIC program already loaded in memory. Any lines in the program in memory which have corresponding lines in the disk file will be over-written during the MERGE, the remaining lines being interleaved. The effect is identical to typing the contents of the disk file on the keyboard.

Example:

MERGE "SECOND:2"

will insert the contents of SECOND.BAS on the disk in drive 2 into the program already in memory.

---

## OPEN

Format: OPEN mode,#stream,filespec

Default: file extension = .DAT

OPEN can be used to allocate a stream number to a disk file, and perform checks or other actions depending on the mode. With the exception that null filenames are not permitted, the format is the same as that used with cassette files. The permitted modes and actions are:-

Mode	Name	Check/Action
"I"	Input	The file must exist, but access is not restricted to read.
"O"	Output	A new file is created of zero length, with backup facilities as provided by the CREATE command. (Similar operation to cassette).
"A"	Append	An existing file will be opened, or a file created and opened.
"E"	Empty	Any existing file is killed and a new zero length file created.

Files explicitly opened may also be used by FREAD etc. in the usual way.

Example:

```
OPEN "E",#4,"FILE1"
```

will delete the file FILE1.DAT, if it exists, and create a zero length file, associating it with stream 4.

---

## PRINT

Format: PRINT [#stream,][USING string:][output list]

Defaults: stream = 0

PRINT can be used to output to a disk file in the same way as a cassette file. Note that the file must be associated with a stream through an OPEN statement, and that terminators are not normally written between the variables output. If CMFLG (location 2035 or \$7F3) is set to the stream number of the file, then 'cassette mode' is enabled and 'end of line' characters are inserted between variables output.

Example:

```
PRINT #4,variable list
```

will output the data to the file opened as stream 4.

## PROTECT

Format: PROTECT [option] filespec

Default: option = ON

The PROTECT command is used alter the write/delete protect bit in the status byte of the directory entry for the file. The option ON is used to set the bit, while the option OFF will clear the bit. Files that are protected cannot be written to or killed and have a lower case 'p' (inverse video for the standard DRAGON screen) against the name in a directory listing.

Example:

PROTECT OFF "PROG5.BAS"

will clear the write/delete protection bit for the file PROG5.BAS

---

## RENAME

Format: RENAME filespec1 TO filespec2  
RENAME #drive TO diskname[,disk number]

RENAME will change the name of a file by altering the directory entry for the file and without copying data. It can also be used to change the name of a disk and optionally change the disk number without overwriting files. Note that for files, both file specifications **must** refer to the same disk drive, while the disk number must be in the range 1 to 255.

Examples:

RENAME "OLDNAME.BAS" TO "NEWNAME.BAS"

will change the directory entry for OLDNAME.BAS on the disk in the default drive (DEFD) to contain the name NEWNAME.BAS

RENAME #3 TO "NEWNAME.DSK",151

will rename the disk in drive 3 to NEWNAME.DSK and alter the number to 151.

---

## RESTORE

Format: RESTORE [line number]

Default: line number = first program line

RESTORE is used to set the data pointer used by the READ command to the specified program line. The next READ statement will take the data from the first DATA statement occurring on or after the line given. Note that the line specified need not contain a DATA statement, but must exist. Line numbers referred to by a RESTORE will be renumbered by the RENUM command.

Example:

RESTORE 20

will set the data pointer for the READ command to line 20 of the program.



## RUN

Format: RUN [filespec[,number]]

Default: File extension = .BAS  
Number = first line (BASIC)  
          default load address (Binary)

The RUN command is used to combine the load and run/execute procedures for a BASIC or Binary program in a disk file. The optional parameter **number** is the line number at which the program is to start running for BASIC, or the new load address (see LOAD) for a binary program.

Examples:

RUN "MCPROG.BAK"

will load the program MCPROG.BAK from the disk in the default drive and start running it from the first line (if a BASIC program) or the entry address (if machine code).

RUN "2:GRAPHS.BIN",&H4000

will load the binary program GRAPHS.BIN from the disk in drive 2, starting at memory location 4000 (hex). The relative entry position is calculated from the file header and added to 16384 to give the new entry address. The program is entered at this calculated address.

---

## SAVE

Format: SAVE filespec[,start.end.entry]

Default: File extension = .BAS (BASIC)  
          .BIN (Binary)

The SAVE command is used to write a BASIC program, or any area of memory, to disk. The selection is made by command syntax. If no memory details are supplied, the BASIC program area is saved as a file of type 1, otherwise the memory block (graphics, machine code program etc.) is saved as a file of type 2. The SAVE command cannot be used to generate a type 3 segmented file.

Examples:

SAVE "MONEY"

will copy the BASIC program in memory to a file called MONEY.BAS on the disk in the default drive (DEFD)

SAVE "GRAPH:3",1024,1536,0

will save the memory area from 1024 to 1535 inclusive (ie the text screen) as the file GRAPH.BIN on the disk in drive 3. The entry address used by EXEC will be set to 0.

## SREAD

Format: SREAD drive,track,sector,string1,string2

SREAD is used to transfer a single 256 byte sector to two strings, each containing 128 bytes. The first 128 bytes are transferred to the first string and the last 128 bytes to the second string. Note that **all** the characters are transferred, even if these are mainly CHR\$(0) characters.

Example:

SREAD 3,20,1,A\$,B\$

will read sector 1 of directory track 20 on the disk in drive 3, the first 128 bytes being written to string A\$ and the remaining 128 bytes written to string B\$.

---

## SWAP

Format: SWAP variable1,variable2

This command is used to exchange the contents of two numeric or two string variables or array elements.

Examples:

SWAP A(1),A(5)

will transfer the value of A(1) to A(5) and the value of A(5) to A(1)

SWAP A\$,B\$

will exchange the strings associated with A\$ and B\$.

---

## SWRITE

Format: SWRITE drive,track,sector,string1,string2

SWRITE is used to write a single sector of a disk. The 256 bytes required are taken from the two strings specified, the first 128 bytes from the first string and the second 128 bytes from the second string. If either string is less than 128 bytes, CHR\$(0) characters are written to the sector following the string to fill the remainder of the 128 character positions. If a string is longer than 128 characters, only the first 128 characters of that string are used.

Example:

SWRITE 2,15,8,S1\$,S2\$

will write the contents of the strings S1\$ and S2\$ to sector 8 of track 15 of the disk in drive 2.

## VERIFY

Format: VERIFY [option]

Defaults: option = ON

This command is used to turn ON or OFF disk verification. Note that this only affects file sectors, as the directory tracks are **always** verified, and that the sector is only checked for transfer errors. Data read from the disk is **not** compared with the original memory content.

Examples:

VERIFY

will turn on the disk write verification

VERIFY OFF

will turn off verification.

---

## WAIT

Format: WAIT [delay]

The command WAIT is used to delay the execution of the BASIC statement following the WAIT statement for the specified number of milliseconds.

Example:

WAIT 1000

will cause the program to loop for 1 second before continuing at the next program statement. Note that <SHIFT> + <@> can be used to extend the wait indefinitely.



## Appendix 2

### Facilities for Assembler Programmers

The routines specified in this Appendix provide all the disk access procedures needed for manipulating discs and files from assembler programs. Each routine is headed by its entry table address and function. The routines are called by means of an indirect subroutine jump :-

eg JSR [SC004] calls the basic operation processing routine

Each routine specification defines:-

- the registers/memory locations that need to be set on entry
- the registers/memory locations which, on exit, always contain particular new data
- the registers which are altered, but which have no particular content on exit

Definitions:

- Absolute Sector:** The sector on the disk numbered from **zero** starting at track 0 sector 1
- Logical Sector:** The sector of the file numbered from **zero** as the first file sector
- Physical Sector:** The sector on the disk referred to by track and sector number within the track

Relationship between sectors:

$$\begin{aligned} \text{Absolute Sector Number} &= \text{Number of sectors per track} * \text{Track Number} + \text{Sector on Track} - 1 \\ &= \text{Absolute Sector at start of file block containing sector} + \text{Logical Sector} - \text{Sectors in preceding file blocks} \end{aligned}$$

## [SC004]

**Basic disk operation processor.** This routine provides the controller level facilities for performing disk input and output. It is used extensively by DOSplus to provide the actual data transfer mechanism for commands relating to the disk. Note that sector read/write operations do not include a SEEK and that head movement may be minimised by issuing a READ ADDRESS (code 6 or 8) to test for drive ready before issuing a RESTORE (code 0), SEEK (code 1) or WRITE TRACK (code 5). Note also that a SEEK specifying track zero is performed as a RESTORE.

Entry: Operation block set up (see [SC006]). DSKCOM values for functions are:-

- 0 = Restore drive to track zero
- 1 = Seek track
- 2 = Read sector
- 3 = Write sector, verify if flag is ON
- 4 = Write sector, no verify
- 5 = Write track (used by initialise)
- 6 = Read address off track
- 7 = Read sector to verify, retain the first two bytes in memory locations 79 and 80.  
The remaining 254 bytes will be discarded.
- 8 = Read address off track into memory locations 18 to 23

Exit: B = Error code (0 if no error)  
C flag set if an error occurs

## [§C006]

### Operation block address for [§C004] operations

- +0 = Operation code for [§C004] routine (DSKCOM)
  - +1 = Drive number (DSKDRV)
  - +2 = Track number (DSKTRA)
  - +3 = Sector number (DSKSEC)
  - +4,5 = Buffer address (DSKBUF)
  - +6 = Status (after operation) (DSKSTA)
  - +7 = File control block number (DSKCBK)
  - +8 = Number of bytes in buffer for read/write  
or number of tracks for DSKINIT
  - +9 = Number of bytes for buffer transfer for read/write  
or side formatting DSKINIT
  - +10 = Record flag, 0 → variable, -1 → fixed for read/write with FOR  
or number of sides for DSKINIT
  - +11 = Read(0)/write(1)/verify(-1) flag
  - +12 = Motor time out flag, 0 → check for time out (DSKMTO)
  - +13 = Close mode (Used by BASIC) (CLFLG)
- 

## [§C008]

**Copy file details and validate.** This routine provides the necessary facilities for validating a file specification, adding a default drive number or file extension if they are not included in the specification.

Entry: B = File specification length (<15 characters)  
X = Address of file specification  
Y = Address of default file extension

Exit: B = Error code (0 if no error)  
\$650-\$657 = Filename  
\$658-\$65A = File extension  
\$65B = Drive number  
Z flag clear if an error occurs

Altered: A,X,U

---

## [§C00A]

**Get directory entry and copy to control block.** Following a call to [§C008], a call to this routine will compare the validated file specification with that in each file control block, starting with block 0. If no match is found, a free block is obtained (or ?TF ERROR occurs) and the file specification entered, with NE status. The directory is searched for the file and, if found, the file control block is loaded with the file detail, including the file status.

Entry: File specification in \$650-\$65B

Exit: B = Error code (0 if no error)  
X = Address of byte +12 of control block  
A,DSKCBK = Control block number  
DSKDRV = Drive number  
Z flag clear if an error occurs

Altered: Y,U



## [§C00C]

**Create directory entry.** This is used to create a directory entry for the specified file, renaming an existing file and deleting an existing .BAK file if required.

Entry: A = File control block number (control block already set up)

Exit: B = Error code (0 if no error)  
Z flag clear if an error occurs

Altered: A,X,U

---

## [§C00E]

**Get file length.** Load the logical sector number and number of bytes used for last file block into the control block.

Entry: A = Control block number for file

Exit: B = Error code (0 if no error)  
U = number of next sector to be written  
A = number of bytes used in the sector  
Z flag clear if an error occurs

Altered: Y

---

## [§C010]

**Close all files on a drive.** Find a control block for the drive specified and call [§C012] to close the file. The drive status is set to 'unaccessed'.

Entry: DSKDRV = Drive number

Exit: B = Error code (0 if no error)  
Z flag clear if an error occurs

Altered: A,X,U

---

## [§C012]

**Close a file.** Release the file control block and if the last file open on the drive, free the unused sectors for the drive and update the bit map on track 20.

Entry: A = Control block number of the file to be closed

Exit: B = Error code (0 if no error)  
Z flag clear if an error occurs

Altered: A,X,U

---

## [SC014]

**Load a file block into memory.** Up to 64K bytes may be loaded into memory from a disk file. The start byte number + number of bytes to load must not exceed the file length. The file control block number will have been obtained through a call to [SC00A].

Entry: A = File control block number for file  
U,B = 24 bit byte number from which loading is to start, B is  
least significant 8 bits  
X = Memory start address  
Y = Number of bytes to load

Exit: B = Error code (0 if no error)  
Z flag clear if an error occurs

Altered: A,Y,U

---

## [SC016]

**Write buffer to file and verify if flag set.** This routine will write up to 64K bytes of data from memory into a disk file. The start byte number must not be greater than the file length.

Entry: A = File control block number  
U = Number of bytes to write  
X = Memory start address  
Y,B = 24 bit byte number from which writing is to start, B is  
least significant 8 bits

Exit: B = Error code (0 if no error)  
Z flag clear if an error occurs

Altered: A,Y,U

---

## [SC018]

**Count number of free sectors on disk.**

Entry: DSKDRV = Drive number

Exit: B = Error code (0 if no error)  
X = Count of sectors  
Z flag clear if an error occurs

Altered: A,Y

## [SC01A]

**Kill a file and free sectors for re-use.** The file referred to by the specified control block will be removed from the directory and the sectors allocated to the file returned for re-use. The file control block is altered to NE status and not released.

Entry: A = Number of file control block containing file details

Exit: B = Error code (0 if no error)  
Z flag clear if an error occurs

Altered: X

---

## [SC01C]

**Set file protection.** The delete/write protect bit in the directory entry for the specified file will be set/cleared as defined by the B register and the control block updated.

Entry: A = Number of file control block holding file details  
B = 0 (protect off), #0 (protect on)

Exit: B = Error code (0 if no error)  
X = Directory record address (if no error)  
Y = Buffer details block address (if no error)  
Z flag clear if an error occurs

Altered: A,X,Y

---

## [SC01E]

**Rename a file.** The file detail in the directory record and in the control block will be changed to the new file specification. Note that no checks are performed to see that the drive numbers in the two file specifications refer to the same drive, the drive number in \$65B is ignored. The file control block is not released at the end of this routine. To avoid duplication of filenames, [SC00A] should be used to ensure a file does not already exist corresponding to the new file specification, before calling this routine.

Entry: A = File control block number for old file  
\$650-\$65A = New file details

Exit: B = Error code (0 if no error)  
Y = Buffer details block address (if no error)  
Z flag clear if an error occurs

Altered: A,X,Y,U



## [§C020]

**Get directory record.** This will get the specified directory record, in the range 0 to 159 (0 to 79 in single density), into memory.

Entry: B = Directory record entry number (starting from 0)

Exit: B = Error code (0 if no error)  
X = Points to required entry  
U = Disk buffer details pointer  
Z flag clear if an error occurs

Altered: A,Y

---

## [§C022]

**Find a free buffer and read absolute sector.** This routine is used to get a particular sector off disk, without using the file details in the directory. If necessary, a buffer will be cleared to allow the data to be read into memory.

Entry: Y = Absolute sector number  
DSKDRV = Drive number

Exit: B = Error code (0 if no error)  
X = Disk buffer details pointer  
Z flag clear if an error occurs

---

## [§C024]

**Copy directory sectors from track 20 to track 16.** This routine is called to backup the directory on track 16 for standard DRAGONDOS format disks, after clearing the disk buffers. DOSplus standard disks will generate an error.

Exit: B = Error code (0 if no error)  
Z flag clear if an error occurs

Altered: X,Y,U,DSKTRA,DSKSEC

---

## [§C026]

**Read absolute sector.** The absolute sector specified in the Y register is transferred to the buffer.

Entry: X = Buffer address  
Y = Absolute sector number  
DSKDRV = Drive number

Exit: B = Error code (0 if no error)  
Z flag clear if an error occurs

## [§C028]

**Write absolute sector.** The buffer contents are written to the absolute sector specified in the Y register. No read-after-write verify is performed.

Entry: X = Buffer address  
Y = Absolute sector number  
DSKDRV = Drive number

Exit: B = Error code (0 if no error)  
Z flag clear if an error occurs

---

## [§C02A]

**Verify absolute sector.** The specified absolute sector is read to check for transfer errors and the first two bytes of the sector are stored in memory locations 79 and 80.

Entry: Y = Absolute sector number  
DSKDRV = Drive number

Exit: B = Error code (0 if no error)  
Z flag clear if an error occurs

---

## [§C02C]

**Format Disk.** The disk in the specified drive is initialised by writing track and sector information to the disk, reading every sector to check for transfer errors, and writing a null directory to the first 18 sectors (10 in single density) of track 20. Before writing to the disk, a 'SURE?' query is output, and formatting may be aborted by pressing <BREAK> at this time.

Entry: B = Number of tracks (40 or 80)  
DSKDRV = Drive number  
[§C006]+10 = 0 for 40 or 80 track single sided disk  
1 for 40 or 80 track double sided disk  
\$650 = Zero if no disk name (or call [§C008] first)

Exit: B = Error code (0 if no error)  
Z flag clear if an error occurs  
Disk workspace is reset

---

**Note:** Non-standard disks may be formatted (eg 35 tracks), but the number of sectors is always 18 (10 single density) single sided or 36 (20 single density) double sided. If the number of tracks is less than 21, an error will occur when an attempt is made to write the empty directory, though the disk will have formatted correctly if track 0 has been checked.

---

## [§C02E]

**Base address of table of disk error code character pairs.** The two character error code may be obtained by subtracting 128 from the code number and adding the address at §C02E. Note that this only applies to disk errors.

## **Appendix 3**

# **DOSplus Hooks and Workspace**



## Hooks

Hook Address	Called From	Function
015E	B828 *	Open drive or file
0161	B7EB *	Check I/O device number
0164	B595	Return device parameters
0167	B54A *	Character output
016A	B50A *	Character input
016D	B623 *	Check device is open for input
0170	B63C *	Check device is open for output
0173	B65C *	Close all devices and files
0176	B663 *	Close a single device or file
0179	84DE *	About to deal with first character of statement
017C	8792	Disc file item scanner
017F	B77B	Poll for BREAK or special keys
0182	B5C6 *	Read a line of input
0185	B6FD	Finish loading ASCII program
0188	B800 *	End of file (EOF) function
018B	8954	Evaluate an expression
018E	8344	User error trap
0191	8347 *	System error trap
0194	85A5 *	RUN statement
0197	8424	CLEAR statement
&	8C80	String copy check
019A	849F	Fetch next statement
019D	86D7	LET string copy check
01A0	850F *	Function assignment validation
&	9EEB *	RENUM statement, token processing
&	AAF7	PUT/GET statement
&	BA5F	CLS statement
01A3	8F67	Compress BASIC line for storage
01A6	8F08	Expand BASIC line for listing

\* Used by DOSplus

## Disc Workspace

Address Hex	Len Dec	Contents
0600	1536 5	Temporary storage
0605	1541 1	Count until motor off (20mS IRQ)
0606	1542 1	Side select/head load timing delay (DOSplus)
		Side compare/head load timing delay (DOSplus DELTA)
0607	1543 1	Copy of drive select latch
0608	1544 1	Verify flag, (zero = on)
060A	1546 1	Default drive number (DEFD)
060B	1547 2	Address of FWRITE buffer
060D	1549 2	Current line number for AUTO
060F	1551 2	Current increment for AUTO
0611	1553 1	RUN/LOAD flag
0612	1554 1	FREAD/FLREAD flag
0613	1555 1	AUTO flag, 0=off, -1=on
0614	1556 1	ERROR flag, 0=off, -1=on
0615	1557 2	ERROR destination line number
0617	1559 2	Line number in ERROR
0619	1561 1	ERROR type
061A	1562 2	Address of start of statement in error
061C	1564 6	Drive 1 details
0622	1570 6	Drive 2 details
0628	1576 6	Drive 3 details
062E	1582 6	Drive 4 details

Address		Len	Contents
Hex	Dec		
0634	1588	7	Disc buffer 1 details
063B	1595	7	Disc buffer 2 details
0642	1602	7	Disc buffer 3 details
0649	1609	7	Disc buffer 4 details
0650	1616	51	Current drive information
0683	1667	20	USR vector table (moved from \$0134)
0697	1687	24	Drive descriptor table
0697	1687	4	Flag 0 → disc not yet accessed
069B	1691	4	Track number of head position
069F	1695	4	Drive step rate (20mS default)
06A3	1699	4	Number of tracks on drive
06A7	1703	4	Number of sectors on drive
06AB	1707	4	Reserved for future use (density select)
06AF	1711	14	Not used
06BD	1725	310	10 file control blocks of 31 bytes
07F3	2035	1	Cassette mode flag (CMFLG)
07F4	2036	12	Not used
0800	2048	1024	4 disc buffers of 256 bytes each

## Disk control blocks

### Drive Details, 4x6 bytes from \$061C

- +0/1 Absolute sector containing first directory sector on current disk
- +2 Number of sectors allocated but not used
- +3/4 First absolute sector in free block
- +5 Count of files open on drive

### Disk Buffer Details, 4x7 bytes from \$0634

- +0/1 Absolute sector in buffer
- +2 Flag, 0 = not in use, \$19 = locked, <0 = modified, write before re-use
- +3 Drive number containing sector
- +4 Buffer allocation number (1 = first allocated)
- +5/6 Buffer address

### Drive Descriptor, 4x6 bytes interleaved from \$0697

- +0 Access flag, 0 → disk not accessed
- +4 Head position (track number)
- +8 Drive step rate (default 2) (0=6mS, 1=12mS, 2=20mS, 3=30mS)
- +12 Number of tracks on drive (from disk directory)
- +16 Number of sectors per track (from disk directory)
- +20 Reserved (Density flag)

### File Control Blocks, 10x31 bytes from \$06BD

- +0/7 File name (+0 = 0 → not in use)
- +8/10 File extension
- +11 Drive number
- +12/14 Read pointer
- +15 File attributes
- +16/18 File write pointer (last byte written)
- +19/23 File block details (1)
  - +19/20 Number of sectors in file before block
  - +21/22 Absolute sector at start of block
  - +23 Number of consecutive sectors in block
- +24/28 File block details (2)
  - +24/25 Number of sectors in file before block
  - +26/27 Absolute sector at start of block
  - +28 Number of consecutive sectors in block
- +29 Master directory entry record number
- +30 Current continuation directory entry record number

## Disk I/O Addresses

Disk Controller Chip (2797 DOSplus, 2791 DOSplus DELTA)

Address		Function
Hex	Dec	
FF40	65344	Command (write) and Status (read) Registers
FF41	65345	Track Register
FF42	65346	Sector Register
FF43	65347	Data Register
FF44	65348	Drive Select Latch (DOSplus DELTA)
FF48	65352	Drive Select Latch (DOSplus)

### Latch Bits

Bit	DOSplus Function	DOSplus DELTA function
0,1	Drive Number	Drive Number
2	Motor on(1)/off(0)	Side Select
3	Single(1)/Double(0) Density Select	5¼"(0)/8"(1) select
4	Write Precompensation Enable	Single(0)/Double(1) Density Select
5	NMI Enable	Not Used
6,7	Not Used	Not Used

### Notes:

1. The 2791 controller chip contains an inverting data bus interface.
2. The DELTA system uses a monostable for motor control, triggered by accessing the controller chip.



## Appendix 4

### DOSplus File Header Format Directory Sector and Record Format

## BASIC and Machine Code Headers

Byte Number

+0	+1	+2,+3	+4,+5	+6,+7	+8
\$55					\$AA
Type		Load Address	Length	Entry Address	

Type      File contents

- 01      BASIC
- 02      Machine Code
- 03      Segmented Machine Code

Load Address is the address from which data started to be saved

Length is the total number of bytes of program saved (for current segment if type 3)

Entry Address is as specified for Machine Code, or \$8B8D (?FC ERROR) for BASIC files

## Directory Sectors

Sector 1 Byte Allocation

Byte (DD)	Contents	Byte (SD)
0 to 89	bit map for tracks 0 to 39 single sided, 0 to 19 double sided	0 to 49
90 to 179	zero or bit map for tracks 40 to 79 single sided, 20 to 39 double sided	50 to 99
—	zero or bit map for tracks 40 to 71 double sided	100 to 179
180 to 188	zero (reserved)	180 to 188
189 to 238	unused	189 to 238
239	Disk number	239
240 to 250	Disk name	240 to 250
251	Single/Dual directory track flag (bit 7 set for single directory track)	251
252	number of tracks (40/80)	252
253	number of sectors/track (18/36 or 10/20)	253
254	complement of 252 (215/175)	254
255	complement of 253 (237/219 or 245/235)	255

Sector 2 Byte Allocation

Byte (DD)	Contents	Byte (SD)
0 to 143	zero or bit map for tracks 40 to 71 double sided	—
144 to 179	zero or bit map for tracks 72 to 79 double sided	0 to 19
180 to 188	zero or bit map for tracks 80 to 81 double sided	20 to 24
—	reserved (zero)	25 to 188
189 to 255	unused	189 to 255

Sectors 3 to 18 Byte Allocation

Byte	Contents
0 to 249	10 by 25 byte file records
250 to 255	unused

## File Record Format

### Standard Entry

+0	+1 to +8	+9 to +11	+12 to +23	+24
Attribute	File Name	Type	Block Info	Qualifier

### Continuation Entry

+0	+1 to +21	+22	+23	+24
		00	00	
Attribute	Block Information			Qualifier

### Attribute Byte

#### Bit    Function (if bit set)

- 0    No file name details in this record
- 1    File protected
- 2    ?
- 3    No more entries follow in directory
- 4    ?
- 5    File details continued in continuation record
- 6    ?
- 7    Record not allocated

### Block Information

Three byte field for each contiguous block (up to 255 sectors) of the file.

#### Byte    Function

- 0,1    Absolute sector number of start of block
- 2    Number of consecutive sectors in block

### Qualifier Byte

Attribute byte Bit 5 affects function as follows

#### Bit 5    Qualifier Byte Function

- 0    Number of bytes in last sector of file
- 1    Absolute record number of continuation file record (starting at zero)





## Appendix 5

### ERROR CODES

## Error Codes - Numeric Order

Value Hex Dec	Code	Meaning
00 0	NF	NEXT without FOR
02 2	SN	Statement syntax error
04 4	RG	RETURN without GOSUB
06 6	OD	Out of DATA in READ
08 8	FC	Function call error
0A 10	OV	Overflow in calculation
0C 12	OM	Out of memory
0E 14	UL	Undefined line number
10 16	BS	Bad array subscript
12 18	DD	Array dimensioned twice
14 20	/0	Attempt to divide by zero
16 22	ID	Illegal direct statement
18 24	TM	Variable/constant type mismatch
1A 26	OS	Out of string space
1C 28	LS	Long string > 255 characters
1E 30	ST	String formula too complex
20 32	CN	Cannot continue after BREAK
22 34	UF	Undefined function
24 36	FD	Faulty file data
26 38	AO	I/O device already open
28 40	DN	Device number outside limits
2A 42	IO	I/O error
2C 44	FM	File mode error
2E 46	NO	File not open
30 48	IE	End of file reached on input
32 50	DS	Direct statement error
34 52	NE	DLOAD file does not exist (D64)
80 128	NR	Disk not ready
82 130	SK	Track seek error
84 132	WP	Disk write protected
86 134	RT	Record type incorrect
87 135	TR	Track not available on the disk
88 136	RF	Record not found - requested sector not on disk
8A 138	CC	Cyclic redundancy check error
8C 140	LD	Lost data - data not transferred to/from disk
8D 141	DB	No duplicate directory track
8E 142	BT	BOOT failure
90 144	IV	Invalid directory
92 146	FD	Directory full
94 148	DF	Disk full
96 150	FS	File specification faulty
98 152	PT	File write (delete) protected
9A 154	PE	Attempt to read past end of file
9C 156	FF	File not found
9E 158	FE	File already exists
9F 159	EN	Entry number above directory limit
A0 160	NE	Disk file does not exist
A2 162	TF	Attempt to open too many disk files (>10)
A4 164	PR	Parameter error
A6 166	??	Unknown error

Error codes \$80 to \$A6 are DOSplus additions



## Error Codes - Alphabetic Order

Code	Value		Meaning
	Hex	Dec	
/0	14	20	Attempt to divide by zero
??	A6	166	Unknown error
AO	26	38	I/O device already open
BS	10	16	Bad array subscript
BT	8E	142	BOOT failure
CC	8A	138	Cyclic redundancy check error
CN	20	32	Cannot continue after BREAK
DB	8D	141	No duplicate directory track
DD	12	18	Array dimensioned twice
DF	94	148	Disk full
DN	28	40	Device number outside limits
DS	32	50	Direct statement error
EN	9F	159	Entry number above directory limit
FC	08	8	Function call error
FD	24	36	Faulty file data
or	92	146	Directory full
FE	9E	158	File already exists
FF	9C	156	File not found
FM	2C	44	File mode error
FS	96	150	File specification or diskname faulty
ID	16	22	Illegal direct statement
IE	30	48	End of input file reached
IO	2A	42	I/O error
IV	90	144	Invalid directory
LD	8C	140	Lost data - data not transferred to/from disk
LS	1C	28	Long string > 255 characters
NE	34	52	DLOAD file does not exist (D64)
or	A0	160	Disk file does not exist
NF	00	0	NEXT without FOR
NO	2E	46	File not open
NR	80	128	Disk not ready
OD	06	6	Out of DATA in READ
OM	0C	12	Out of memory
OS	1A	26	Out of string space
OV	0A	10	Overflow in calculation
PE	9A	154	Attempt to read past end of file
PR	A4	164	Parameter error
PT	98	152	File write (delete) protected
RF	88	136	Record not found - requested sector not on disk
RG	04	4	RETURN without GOSUB
RT	86	134	Record type incorrect
SK	82	130	Track seek error
SN	02	2	Statement syntax error
ST	1E	30	String formula too complex
TF	A2	162	Attempt to open too many disk files (>10)
TM	18	24	Variable/constant type mismatch
TR	87	135	Track not available on the disk
UF	22	34	Undefined function
UL	0E	14	Undefined line number
WP	84	132	Disk write protected

Error codes \$80 to \$A6 are DOSplus additions



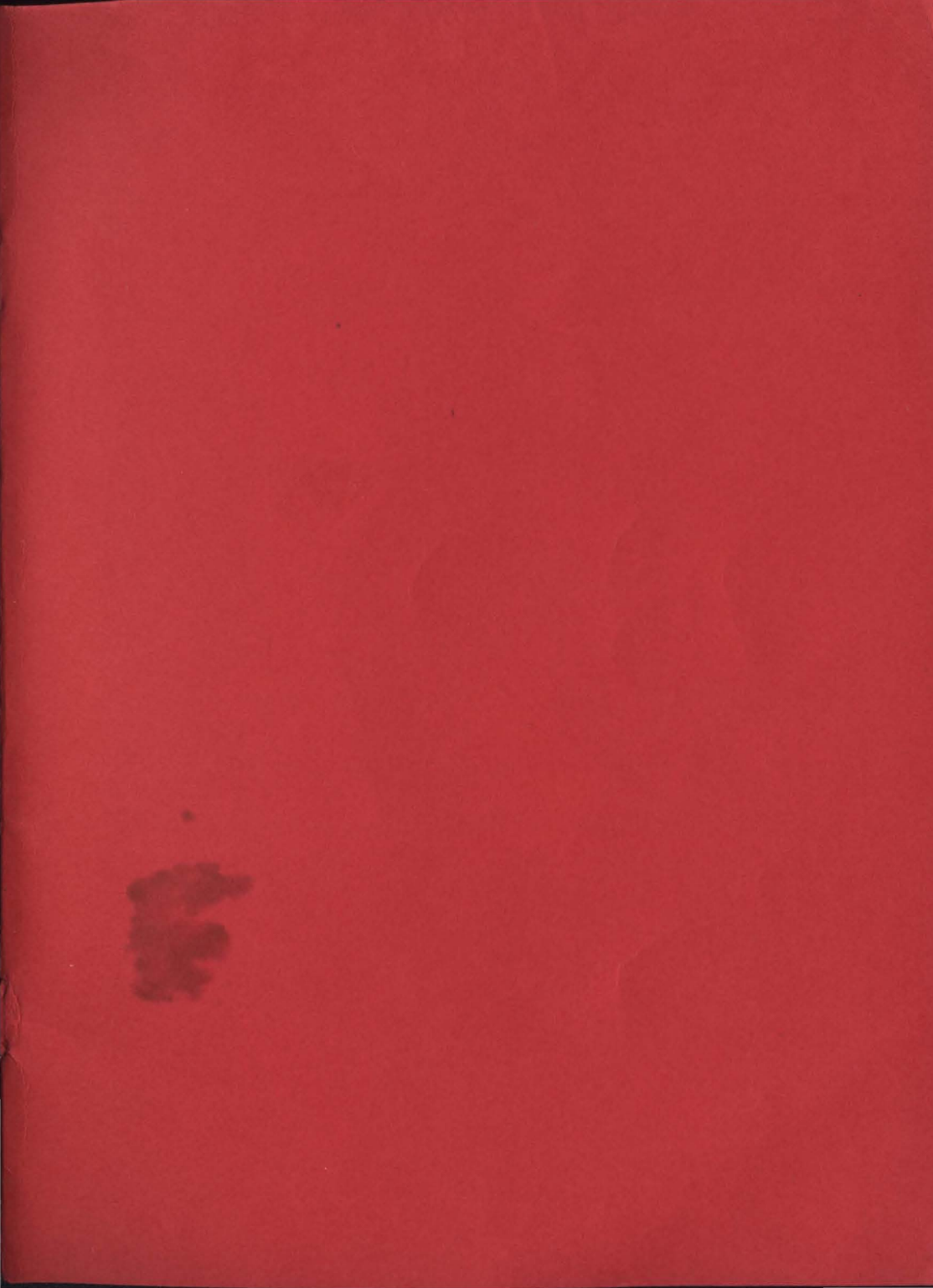
## **Appendix 6**

### **COMMAND SUMMARY**



## Command Summary

Command	Parameters	Page
AUTO	[start [,increment] ]	22
BACKUP	[source [TO destination [,sides [,tracks] ] ] ]	22
BACKUP DIR	[-] [drive]	23
BEEP	[number]	23
BOOT	[drive]	23
CHAIN	filespec [,start]	24
CLOSE	[stream [,stream [, ...] ] ]	24
COPY	[/] source filespec TO destination filespec	24
CREATE	filespec [,length]	25
DIR	[/][drive]	25
DRIVE	drive	26
DSKINIT	[drive [,sides [,tracks] ] ]	26
EOF	(filespec)	27
ERL		27
ERR		27
ERROR GOTO	line number	28
FLREAD	filespec [,FROM start][,FOR length] ; string	28
FREAD	filespec [,FROM start][,FOR length] ; variables	29
FRES		29
FREE	[drive]	30
FROM	filespec	30
FWRITE	filespec [,FROM start][,FOR length];[USING format;][variable list]	30
HIMEM		31
INPUT	#stream, variable list	31
KILL	filespec / [drive]	31
LINE INPUT	#stream, string	32
LOAD	filespec [,binary load address]	32
LOC	filespec	33
LOF	filespec	33
MERGE	filespec	33
OPEN	mode,#stream,filespec	34
PRINT	[#stream,][USING string;][output list]	34
PROTECT	[option] filespec	35
RENAME	filespec1 TO filespec2 #drive TO diskname[,disk number]	35
RESTORE	[line number]	35
RUN	[filespec[,number]]	36
SAVE	filespec[,start,end,entry]	36
SREAD	drive,track,sector,string1,string2	37
SWAP	variable1,variable2	37
SWRITE	drive,track,sector,string1,string2	37
VERIFY	[option]	38
WAIT	[delay]	38







## ERRATA

The following corrections should be applied to your manual.

\*\*\* Page 16 - section 6.2 should read:-

### 6.2 Text Command Files

It is possible with DOSplus to read commands from a disk file instead of the keyboard. This is accomplished with the FROM command. For example,

#### FROM "COMLIST"

will take a single line of input from the file **COMLIST.CMD** instead of the keyboard (**.CMD** is the default extension). This is not as restrictive as it may seem, as the line may contain a further FROM statement. The line to be obeyed must not be more than 250 characters long, and if less than 250 characters, the line should contain a CHR\$(0) as the last character. To set up a file, use a program such as the following:

```
10 LINE INPUT CMD$ : FWRITE "STARTUP.CMD"; CMD$+CHR$(0) : CLOSE
```

\*\*\* Page 20, line -4 should read

Note that BOOT may be automatically entered at power on or any cold reset by pressing **Y**.

\*\*\* page 30 - FROM details should read:-

## FROM

Format: FROM filespec

Default: file extension = .CMD

FROM is used to obey a line of up to 250 characters from a command file in preference to a line of keyboard input. A CHR\$(0) character is inserted automatically at character position 251, but lines of less than 250 characters must end with a CHR\$(0) character. There must not be an 'end of line' character before the CHR\$(0) terminator. A single statement only may be obeyed, but this may include a further FROM call. Once loaded into the keyboard buffer, the normal keyboard input rules apply.

Example:

```
FROM "STARTUP"
```

\*\*\* Page 52, last line

250 unused  
251 to 255 as sector 1

## Comments

1. Replacing line 70 of the program in section 6.12 with

```
70 A$ = LEFT$(A$ + STRING$(128,0),123) + "FROM" + CHR$(34)
```

will enable a 'FROM' command to be obeyed instead of 'RUN'.

2. If an ?IV ERROR occurs when trying to recover a corrupted directory, copy track 16 sector 1 or any track 20 directory sector (except sector 2) into track 20 sector 1 and try again.

3. If a disk is being used as a scratch disk to hold files temporarily, then use 'BACKUP DIR' immediately after 'DSKINIT' to copy the empty directory, and 'BACKUP DIR -' may then be used to effectively 'KILL' all files on the disk.

## ERRATUM

The following correction should be applied to your manual.

\*\*\* Page 4 - section 2.5 should read:-

### 2.5 DOSplus Disable

For those occasions where it is necessary to run without a DOS capability (for example: running cassette based games which write to the DOSplus workspace area as mentioned in 2.4 above), it is no longer necessary to remove the cartridge. At any time input is accepted, alter the 'secondary reset vector' by:

**POKE 115,PEEK(115)-4**

and press the reset button. This will perform a partial cold start, retaining the top of memory and string space pointers, but setting the graphics base to 1536 (\$600) and clearing any BASIC program. DOSplus can be re-enabled by performing a full cold start (**POKE 113,0** and press reset) or **POKE 235,0 : EXEC 49154**.



# DOSplus Extensions

## LOAD

Format:    LOAD / filespec

Default:   file extension = .ASC

The LOAD/ command is used to transfer the contents of a data file containing an ASCII dump of a BASIC program into memory from disk, replacing any existing program.

Example:

LOAD / "BASPROG"

will load the BASIC program BASPROG.ASC from the disk in the default drive .

---

## MERGE

Format:    MERGE / filespec

Default:   file extension = .ASC

MERGE/ is used to superimpose a BASIC program held as an ASCII dump in the specified disk file on a BASIC program already loaded in memory. Any lines in the program in memory which have corresponding lines in the disk file will be over-written during the MERGE, the remaining lines being interleaved. The effect is identical to typing the contents of the disk file on the keyboard.

Example:

MERGE / "SECOND:2"

will insert the contents of SECOND.ASC on the disk in drive 2 into the program already in memory.

---

## SAVE

Format:    SAVE / filespec[,list range]

Default:   file extension = .ASC

The SAVE/ command is used to write a BASIC program, or part program, to disk as an ASCII dump. A part file is saved by specifying a 'list range' in the same format as used by the LIST command.

Example:

SAVE / "GRAPH",1024-1536

will save lines 1024 to 1535 inclusive as the file GRAPH.ASC on the disk in drive 3.

## Electronic Author

There are a number of errors in "Electronic Author" which are shown up when it is used with DOSplus. The corrections for these are supplied in a form which will allow Electronic Author to be used with ANY DOS. Load the program and type the changes shown below, then save the program with:-

SAVE "AUTHOR", &H2400, &H3C00, &H2813

1. Files will not load properly, as the "file control block" number is not used correctly. The correction is:-

POKE &H28FE, 82 : POKE &H2904, 82

2. Only files opened on a disk in drive 1 are closed properly. The correction is:-

POKE &H2A4A, 18 : POKE &H2A4B, 18 : POKE &H2A4C, 18 : POKE &H2A4D, 18

3. Directory listing is not handled correctly - files are missed out, spurious characters are output, the listing can be shifted right, long directories are scrolled too quickly and the program is DOS dependent as a routine accessible through the entry table is called directly. The corrections which cure the problems and implement a 28 file paging (press any key to get the next page), are given in the form

address hex hex hex hex .....

The two groups of changes can be implemented by POKEing successive locations.

```
3B95 86 A0 97
3B98 E7 86 1C 97 E9 D6 E8 AD
3BA0 9F C0 20 A6 80 85 08 26
3BA8 3B 85 81 26 1F C6 08 8D
3BB0 23 86 2E BD B5 4A C6 03
3BB8 8D 1A 8E 3B F3 C6 04 8D
3BC0 13 0A E9 26 07 BD A0 EA
3BC8 86 1C 97 E9 0C E8 0A E7
3BD0 27 12 20 C9 A6 80 26 02
3BD8 86 20 BD B5 4A 5A 26 F4
3BE0 39
```

```
3BF3 20 20 20 20
```