



DRAGONDOS COMMANDS

10 ZZ = FREE FORCE GARBAGE COLLECTION
20 REM REST OF CHAINED PROGRAM FOLLOWS

AUTO

The command AUTO produces line numbers for a program automatically.

AUTO
produces lines starting with 100 and incrementing in steps of 10.

AUTO 50,5
produces lines starting with 50 and incrementing in steps of 5.

The AUTO mode is terminated by pressing [ENTER] immediately after a line number has appeared.

AUTO overwrites lines which have the relevant numbers, but leaves others unchanged.

BACKUP

The command BACKUP creates a backup copy of a complete disk (1 or 2 sides).

BACKUP 3 TO 4, 2, 80
creates a backup copy of disk 3 on disk 4 (2 sides, 80 tracks).

If the source and destination disks are to use the same drive, your Dragon will give instructions for inserting the source and destination disks alternately.

BACKUP
creates a backup copy of a source disk on a destination disk, using only drive DEFD, 1 side, 40 tracks.

BEEP

The command BEEP gives an acceptable sounding 'beep'.

BEEP
gives 1 beep.

BEEP 20
gives 20 beeps.

BOOT

The command BOOT loads an operating system from a disk into memory, starting at 9728.

BOOT
loads the system from disk DEFD.

BOOT2
loads the system from disk number 2.

The system is executed using EXEC9730.

CHAIN

The command CHAIN may be used to load and run a program, preserving the values of variables. This is useful when more than one program is to be run on the same set of data.

CHAIN*PROG2*
loads PROG2.BAS from drive DEFD and starts execution at the first statement without initialising variables to zero or strings to null.

CHAIN*PROG3*,50
loads PROG3.BAS from drive DEFD and starts execution at line 50, without initialising variables to zero or strings to null.

CHAIN*2:PROG*
performs the CHAIN operation on program PROG.BAS from the disk on drive number 2.

When using the CHAIN command with programs that use string variables it is recommended that a garbage collection be forced at the start of every CHAINED program. This can be achieved by including a call to the function FREE as the first statement of the program. For example:

CLOSE

The command CLOSE is used to close files which have been opened by commands such as PARITE, FREAD and FLREAD. A maximum of 10 files may be open at any particular time, so it is sometimes necessary to close files before opening others. When a file is closed, its read pointer accessed by function LOC is set to zero.

CLOSE
closes all files on all disks.

CLOSE2
closes all files on the disk on drive 2.

COPY

The command COPY is used to make copies of files. The copy and the original may be on the same disk or on different disks (if you have more than one drive).

COPY*ORIGINAL.BAS* TO *NEWFILE.BAS*
makes NEWFILE.BAS a copy of ORIGINAL.BAS (on the disk on drive DEFD). The filetype (e.g. BAS) must be included in the specification of both files.

COPY*1:FILE.BAK* TO *2:BACKUP.AAA*
makes the file BACKUP.AAA on disk 2 a copy of FILE.BAK on disk 1.

CREATE

The command CREATE may be used to create a datafile.

CREATE*DATAFILE*
creates DATAFILE.DAT with length 0 bytes.

CREATE*DATAFILE*,80
creates DATAFILE.DAT with length 80 bytes.

DIR

The DIR command lists the files on a disk and gives the number of bytes allocated to each file and the number of free bytes on the disk.

DIR
lists the files on the disk on drive number DEFD (as set by DRIVE).

DIR 2
lists the files on the disk on drive number 2.

A typical directory of files is as follows:

DIR			
BASPROG	BAK	1355	
MACHCODE	BAK	3809	
BASPROG	BAS	1855	
DATAFILE	DAT	3000	
MACHCODE	BIN	1009	
165072 FREE BYTES			

In this directory, BASPROG.BAK is a backup version of Basic program BASPROG.BAS, MACHCODE.BAK is a backup version of binary file MACHCODE.BIN, and DATAFILE.DAT is a data file.

DRIVE

The DRIVE command selects the default drive number, DEFD. All commands which do not specify a particular drive will apply to drive DEFD until DRIVE is used again. On power-up, DEFD is set to 1.

DRIVE 3
sets DEFD to 3.

DSKINIT

The DSKINIT command is used to format a new disk and set up a directory of files. No other commands can be applied to a disk until it is formatted.

DSKINIT

will format one side of the disk on drive number DEFD (1 unless altered by DRIVE), setting up 40 tracks. This command may take about a minute to execute.

DSKINIT 3,2,80

will format both sides of the disk on drive 3, setting up 80 tracks per side.

EOF

The function EOF is used to check whether the read pointer is at the end of a file (i.e. whether all records have been read). The program statement

X=EOF("FILE")

will give X the value 0 if the pointer is at the end of FILE.DAT, and the value 1 otherwise. Note that brackets are required for this function.

ERL

When an error has occurred, the function ERL gives the line number in which the error occurred.

After any error in line 50,
PRINT ERL
will return 50.

ERR

When an error has occurred, the function ERR gives the code number of the error. After an IO ERROR (input/output)
PRINT ERR
will return the value 42 (the code for IO).

Error codes are listed at the back of this manual in Appendix 2.

ERROR GOTO

The command ERROR GOTO directs control to a particular line if an error is subsequently detected. After the command

ERROR GOTO5000

has been executed, if any error is detected, control will pass to line 5000. If another ERROR GOTO statement is executed, it will override the previous instruction.

FLREAD

The command FLREAD is used to read a string from a file. Commas and colons are not regarded as terminators.

FLREAD*FILE*; string
reads a string from FILE.DAT, starting at the position of the read pointer for that file (initially at the beginning of the file). The read pointer is updated to the byte following the string read.

FLREAD*FILE*, FROM 100, FOR 80; string
reads a string from FILE.DAT, starting at byte 100. After reading, the read pointer is advanced to byte 180 (100+80).

Any file accessed by FLREAD is left 'open'. A maximum of 10 files may be open at any given time.

FREAD

The command FREAD is used to read records from a file.

FREAD*FILE*; variable list
reads the variable list from FILE.DAT, starting at the position of the read pointer for that file (initially at the beginning of the file). The read pointer is updated to the byte following the final record read.

FREAD*FILE*, FROM 30, FOR 50; variable list
reads the variable list from FILE.DAT, starting at byte 30. After reading, the read pointer is advanced to byte 80 (30+50). Strings are terminated by commas and colons, as well as 'end of line' characters.

Any file accessed by **FREE** is left "open". A maximum of 10 files may be open at any given time.

FREE

The function **FREE** gives the number of free bytes on a disk.

PRINT FREE

gives the number of free bytes on the disk on drive number DEFD.

PRINT FREE2

gives the number of free bytes on the disk on drive number 2.

FREE may be included in programs, e.g.

```
X=FREE1
```

FRES

The function **FRES** gives the number of free bytes available for strings. It is not a string: it has a numeric value. Use of **FRES** forces a "garbage collection".

PRINT FRES

returns the total number of bytes which are free to be allocated to strings, after the strings already present have been packed efficiently.

FWRITE

The command **FWRITE** is used to write records to a file.

```
FWRITE"FILE";variable list
```

writes the variable list in **FILE.DAT**. If **FILE.DAT** does not already exist, it is created, and writing starts at the beginning. If it does exist, writing starts at the end of the file.

```
FWRITE"FILE", FROM 10, FOR 30; variable list
```

writes the variable list in **FILE.DAT**, starting at byte 10 and extending the record length to 30 bytes. The file must already exist and be at least 9 bytes long to make it possible to start at byte 10.

Any file accessed by **FWRITE** is left "open". A maximum of 10 files may be open at any given time.

HIMEM

The function **HIMEM** gives the highest memory location available to Basic. This is altered by the second parameter of a **CLEAR** statement.

PRINT HIMEM

will return 32768, unless **CLEAR** has been used to reserve space at the top of RAM.

KILL

The command **KILL** is used to delete files from a directory, releasing the disk space for other files.

```
KILL"PROG.BAS"
```

deletes the file **PROG.BAS** from the disk on drive number DEFD. The filetype (e.g. **BAS**) must be included in the specification.

```
KILL"2:CODE.BIN"
```

deletes the file **CODE.BIN** from the disk on drive number 2.

LOAD

The command **LOAD** may be used to transfer Basic or Binary files from disk to memory.

```
LOAD"BPORG"
```

will load the Basic program from file **BPORG.BAS** on drive DEFD into memory.

```
LOAD"CODE.BIN"
```

will load the binary code from file **CODE.BIN** into the same area of memory from which it was originally saved, and will set the default **EXEC** address to the entry point specified when the file was saved.

```
LOAD"MACH.BIN",1000
```

will load the binary code from file **MACH.BIN** into memory, starting at address 1000. If **MACH** was originally saved using

```
SAVE"MACH",start,end,entry
```

then the default **EXEC** address will be set to 1000+entry-start. Note that this is a different (and more flexible) convention from that used by **CLOADM**.

```
LOAD"PROG.BAK"
```

will load the file **PROG.BAK**. If it is the backup of a Basic program, it will be loaded into the usual Basic position; if it is a binary file, it will be loaded as **CODE.BIN** above.

```
LOAD"2:PROG"
```

will load **PROG.BAS** from the disk on drive 2.

LOC

The function **LOC** is used to find the position of the read pointer in a file.

```
LOC"DATAFILE"
```

returns the number of the next byte to be read in **DATAFILE.DAT**. For a subsequent "read" command, this will be the default "FROM" parameter.

LOF

The function **LOF** gives the length of a file in bytes.

```
PRINT LOF"CODE.BIN"
```

gives the length of **CODE.BIN** in bytes.

LOF may be used in programs, e.g.

```
X=LOF"PROG.BAS"
```

The filetype (e.g. **BIN** or **BAS**) must be included.

MERGE

The command **MERGE** is used to superimpose a disk file on to a Basic program in memory. The "file on disk" is not affected. The result in memory is a program containing all the line-numbers from the file, and any line-numbers originally in memory which were not in the file. i.e. the file and original program are mixed together, but if there are any line-numbers in common, it is the file versions which are retained.

```
MERGE"PROG"
```

merges the Basic program **PROG.BAS**, on disk DEFD, with any program in memory, overwriting where line-numbers are in common. **MERGE** can only be used with Basic files.

PROTECT

The command **PROTECT** places a file in a "protected" category, so that it cannot be overwritten or deleted.

```
PROTECT ON "PROG.BAS"
```

causes **PROG.BAS** to be protected, so that **KILL"PROG.BAS"** and **SAVE"PROG"** lead to a PT ERROR.

```
PROTECT OFF "PROG.BAS"
```

removes the protection.

A reverse-screen "P" appears against protected files on the file directory.

RENAME

The command **RENAME** is used to change the name of a file on a disk, or to transfer a file from disk to disk.

```
RENAME"OLDNAME.BIN" TO "NEWNAME.BIN"
```

gives the file **OLDNAME.BIN** the new name **NEWNAME.BIN**. Note that the filetype (in this case, **BIN**) must be included in the specification of both names.

RUN

The command **RUN** is used to transfer a Basic file from disk to memory and execute it, starting at the first statement.

```
RUN"PROG1"
```

loads **PROG1.BAS** from drive DEFD and runs the program.

```
RUN"3:FILE.BAK"
```

loads backup Basic program **FILE.BAK** from drive

SAVE

The **SAVE** command may be used to transfer Basic programs or machine code (or other binary sequences) from memory to disk.

```
SAVE"PROG"
```

will create a file **PROG.BAS** on drive DEFD containing the Basic program currently in memory. If a file **PROG.BAS** already exists, it will be transferred to the file **PROG.BAK**. If a file **PROG.BAK** already exists, it will be overwritten.

```
SAVE "CODE",2000,5001,4000
```

will create a file **CODE.BIN** containing the sequence of bytes from start-address 2000 to end-address 5000. When this file is loaded, the default **EXEC** address will be set to 4000. If a file **CODE.BIN** already exists, it will be transferred to **CODE.BAK** etc. The disk drive number may be specified by putting drive: in front of the file-name.

```
SAVE"2:PROG"
```

will save **PROG.BAS** on drive 2.

SREAD

The command **SREAD** is used to read the record contained on a single sector of a disk. The record is 256 bytes, and is read into two strings, e.g. **A\$** and **B\$**, both of length 128 bytes.

```
SREAD 2, 6, 10, A$, B$
```

reads the 256-byte record contained on the 10th sector of the 6th track of the disk on drive 2, placing the first 128 bytes in **A\$**, and the last 128 bytes in **B\$**. Even if they are made up mostly of **CHR\$(0)** characters, both **A\$** and **B\$** will be of length 128 bytes.

SWAP

SWAP exchanges the values of two variables.

```
SWAP X,Y
```

gives **X** the value of **Y** and **Y** the value of **X**.

SWRITE

The command **SWRITE** is used to write a record to a single sector of a disk. The record is assembled in two strings, e.g. **X\$** and **Y\$**, each of maximum length 128 bytes.

```
SWRITE 3, 7, 9, X$,Y$
```

writes the records contained in **X\$** and **Y\$** to the 9th sector of the 7th track of the disk on drive 3. If either **X\$** or **Y\$** is less than 128 bytes, the extra space will be filled with **CHR\$(0)** values.

VERIFY

The commands **VERIFY ON** and **VERIFY OFF** turn the verify procedure on and off. It is normally on (at power-up) and performs a verification on the directory track.

WAIT

The command **WAIT** suspends execution of the program, producing an easily controlled pause.

```
WAIT 5000
```

pauses for 5000 milliseconds (5 seconds).

APPENDIX 2 - ERROR CODES

0	HE	HEAT without HSE	7	AN	anyway error
1	HA	HEAT without HSE	8	AO	Out of data in READ
2	FO	Illegal function call	10	OV	Overflow
12	OU	Out of memory	14	UL	Undefined line
16	HS	Bad subscript	18	NS	Redimension Array
20	JO	Division by zero	22	IL	Illegal direct statement
24	TU	Type mismatch	26	OS	Out of string space
28	LS	String too long	30	ST	String too complex
32	CM	Can't continue	34	OF	
36	SD	String data	38	AS	File already open
40	DN	Drive number	42	IO	Input/output error
44	FM	Using File Mode	46	NS	File not open
48	LE	Input past EOF	50	DS	Direct statements
128	NR	Not Ready	134	AC	Access
130	NP	Not Protected	136	RT	Record type
132	RF	Records Not Found	138	OS	Cyclic Redundancy
140	LD	Last Data	142	BF	Bad
144	IV	Invalid Directory	146	FS	Directory full
148	OF	Open full	150	FS	File full
152	PT	Protection	154	PE	Read past EOF
156	FF	File not found	158	PE	File exists
162	AC	Access denied	162	FF	Too many open