Dragon Data Ltd.

AN INTRODUCTION TO DRAGONDOS

by

ALAN MAYER

A Manual for the Dragon Disk Operating System

First Printing 1983
© 1983. DRAGON Data Limited

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means electronic, mechanical, photocopying, recording, or otherwise; without the prior permission of the publisher.

This book is sold subject to the condition that it shall not by way of trade or otherwise, be lent, resold, hired out, or otherwise circulated without the publisher's prior consent in any form of binding or cover other than that in which it is published and without a similar condition being imposed on the subsequent purchaser.

The author and Dragon Data Limited wish to thank the Statistics Department of University College Swansea for facilities and help in preparing this manual.

CONTENTS

CHAPTER		PAGE
1	Getting the Show on the Road	4
2	There's a File for Everything	6
3	Reading and Writing	13
4	More Power to your Dragon	19

APPENDIX		
1	DRAGONDOS Commands	23
2	Error Codes	59

Page

Chapter 1: Getting the Show on the Road

Connecting your Disk Drive……..……..…….... 4
Inserting the Disk……..……... 4
Which Drive?……..……. 5
Setting Up a New Disk……..…….. 5
Video RAM……..…… 5

Chapter 2: There's A File For Everything

What's On My Disk?……..…… 6
Saving, Loading and Running……..…….. 6
Retrieving Programs from Disk……..…… 7
Saving and Loading Machine Code……...……...…….. 8
Backup Files……...……...…….. 9
Copying and Renaming……..…… 9
Housekeeping……..…… 10
Putting the Pieces Together…… 10
For Your Protection…… 11
A Spare Copy — Just in Case…… 11
Some Useful Functions…….. 12

Chapter 3: Reading and Writing

Data Files……...…….. 13
Simulated Random Access…..…… 16
Too Many Files……..……..17
Is This The End?……..17
Reading and Writing Without Files…….18

Chapter 4: More Power To Your Dragon

Where Have I Put My Memory?…….19
Don't Stop For Errors……...19
On The Third Stroke…….20
Hold On A Moment!……... 20
Fair Exchange……...21
Take A Hundred Lines…… 21
Booting Another DOS……22

CHAPTER1 GETTING THE SHOW ON THE ROAD

Connecting Your Disk Drive

Before turning on the power supply to either Dragon or disk drive, the units should be assembled as follows:—

(a) Connect the ribbon-wire from the disk drive to the DOS (Disk Operating System) cartridge.

(b) Position the disk drive on a horizontal surface near to the Dragon, making sure that the cartridge will reach the Dragon's cartridge port, with plenty of slack in the ribbon-wire.

(c) Insert the DOS cartridge in the cartridge port and push it home, positioning the slack ribbon wire so that it will not be accidentally pulled out.

(d) Connect the disk drive power line to the socket in the back of the disk drive.

(e) The disk drive was supplied with a card in the disk port. This should now be removed.

(f) Ensure that Dragon and TV are properly connected up and turn on the power supplies to all units.

The TV screen should first show the usual copyright messages, then after a few seconds, a special message indicating that the disk drive is attached. (If it does not, switch off and check all the connections.) The message should be followed by OK and the usual cursor.

Inserting the Disk

The correct disk to use is a single sided double density five and a quarter inch mini diskette.

The DOS can support up to four drives with single or double sided disks, but the standard drive unit is single sided. Double sided disks may be used, but only one side will be formatted. Insert the disk into

the disk port, with the open window towards the drive and the small

notch on the left. Secure the disk by pushing down the locking device (which will slide forward a short way when it is secured). To release the disk, press in the locking device and remove the disk.

Which Drive?

If there is more than one drive attached to the DOS, it is important that each command is directed to the correct drive. The DRIVE command may be used to alter the default drive number, DEFD. When **a command does not specify a drive number, DEFD** is assumed. On power-up, DEFD is set to 1.

DRIVE *n*

sets DEFD to *n (n=1,* 2, 3 or 4).

Setting Up a New Disk

When a disk is used for the first time, it must be formatted, or "initialised". This is done by typing the command

DSKINIT *drive, sides, tracks*

and pressing ~NTER]. Parameter *drive* is the drive number (for use when more than one drive is attached), *sides* is the number of sides to be formatted (1 or 2), and *tracks* is the number of tracks per side (40 or 80). The default values for *sides* and *tracks* are respectively 1 and 40. The default for *drive* is DEFD (as set by DRIVE). For the standard Dragon disk drive it is usually sufficient just to type DSKINIT. The disk drive will operate for some time, after which "OK" should appear. If an error message appears, remove and re-insert the disk and try again. Repeated failure to format may be caused by a damaged disk.

Note that a disk should only be formatted the first time it is used. Subsequent formatting will have the .effect of wiping~ all records from the disk.

Video RAM

The DOS cartridge has the effect of moving the video RAM up one page, so that the first page starts at byte 3072 instead of 1536. This can cause problems with some programs designed to be used with the cassette recorder. If they write directly to the area occupied by the DOS, the disk operating system can be corrupted. To restore, just switch the computer off and on again.

CHAPTER2 THERE'S A FILE FOR EVERYTHING

What's on My Disk?

The command DIR *drive*, where *drive* is the number of the required disk drive, will produce a list of all files on the disk on that drive, and will also give the number of free bytes available for new files (this information is also available via the "FREE" function). The default value of *drive* is DEFD(as set by DRIVE).The full title of a file is of the form *drive:filename.filetype*

where *filename* is the user-assigned filename (up to 8 characters, starting with a letter), and *filetype* is a code of up to 3 characters used to distinguish files of the same name. The following three-letter codes are produced automatically to help distinguish files of Basic, machine code etc.

BAS — Basic program
BAK — Backup file
BIN — Binary file, e.g. Machine code
DAT — Data file.

Against each file in the directory will be a number. This is the number of bytes allocated to that file.

DIR 2

gives the directory of files on the disk on drive number 2. If the directory contains a large number of files, [SHIFT] and may be used, as with the LIST command, to pause. Press any key to restart.

Saving, Loading and Running

Saving a Basic Program
If you have a Basic program in memory, it may be saved on disk using the command

SAVE"PROGRAM"

The name "PROGRAM" can be replaced by any name of up to 8 characters, starting with a letter. When you press [ENTER] the drive will operate for a few seconds, and OK will appear on the TV screen. Even programs which take a minute or two to save on tape take only a few seconds on disk.

Now you can use the DIR command to list your files. The file PROGRAM. BAS will be listed. The number against the program indicates the number of bytes taken up by that file.

The filetype BAS is allocated by default, and the BAS, BAK, BIN, DAT nomenclature is designed to help you to keep track of files and to provide a simple backup system. You can, however, invent your own system and use, for example,

```
    SAVE"PROG.A"
or  SAVE"PROG.1A"
```

Retrieving Programs from Disk

If a Basic program called "PROGRAM" has been saved on a disk, it may be loaded into memory using the command

```
LOAD"PROGRAM"
```

If you want to load and run a Basic program, you can use the RUN command, which has been extended by the addition of the DOS cartridge. Without using the LOAD command, you can type

```
RUN"PROGRAM"
```

and the Basic program PROGRAM will be loaded and run. You can, of course, achieve the same result using the two commands

```
LOAD"PROGRAM"
RUN
```

Another way of loading and running is the CHAIN command. This performs in the same way as RUN, except that the values of variables are preserved, instead of being set to zero. The command

```
CHAIN"PROGRAM"
```

will load and run the Basic program PROGRAM, without setting variable values to zero (and strings to null).

As an example of the use of RUN and CHAIN, try typing in the following short programs

```
10 Z=X+Y
20 PRINT "SUM=";Z
SAVE"SUM"
10 Z=X*Y
20 PRINT "PRODUCT=";Z
```

```
SAVE”PRODUCT”
10 Z=(X+Y)12
20 PRINT “AVERAGE=”;Z
SAVE”AVERAGE”
10 INPUT”X”;X
20 INPUT”Y”;Y
SAVE”INPUT”
```

You now have four short files: SUM.BAS, PRODUCT.BAS, AVERAGE.BAS and INPUT.BAS.

You can use

```
RUN”INPUT”
```

to input a value for X and a value for Y. For example

```
X? 10 [ENTER]
Y? 6 [ENTER]
```

then the CHAIN command can be used to work out the sum, product or average.

```
CHAIN”PRODUCT”
```

will give

```
PRODUCT= 60
```

and

```
CHAIN”AVERAGE”
```

will give

```
AVERAGE= 8
```

and so on.

For two numbers, the method is hardly worth bothering about, but for programs involving large amounts of data it can be useful.

CHAIN can be used with a parameter::-

```
CHAIN”PROGRAM”,entry
```

where *entry* is the line-number at which execution is to start.

Saving and Loading Machine Code

Machine code (or any other series of bytes) may be saved on disk using

```
SAVE”CODE” ,start,end,entry
```

where *start* is the starting address of the code in memory, *end* is the addresss of the byte following the last byte of the code in memory and *entry* is the entry point (the default EXEC value). “CODE” may be replaced by any other legal filename.

The file will appear in the directory with the filetype BIN (a binary file). Binary files may be loaded using

LOAD"CODE .BIN",*begin*

where *begin* is the new start address. The default value for *begin* is the start address of the original code (when it was saved). The default EXEC address is calculated relative to *begin.*

Backup Files

When you SAVE a Basic program using a filename which is already in the directory, the old version is transferred to filetype BAK and the new version is recorded in the BAS filetype. Any previous .BAK version is overwritten. e.g. suppose PROGRAM.BAS already exists, and you save a new program using

SAVE"PROGRAM"

There will now be two files:

PROGRAM.BAS
PROGRAM.BAK

PROGRAM.BAS is the second (latest) version; PROGRAM.BAK is the first version.

If you now save another program using

SAVE"PROGRAM"

there will still be two versions recorded, but now PROGRAM.BAS will be the third version and PROGRAM.BAK will be the second version. The first version is no longer recorded.

The backup version may be loaded, using

LOAD"PROGRAM .BAK**"**

(or RUN, or CHAIN) so if you discover that your latest version has errors and you want to go back to your backup version it is always available.

An identical backup system operates for binary files and data files. If two or more files of different filetypes have the same name, the backup files will have the same name and will overwrite each other, so that only the most recent file will have a backup version.

Copying and Renaming

Files may be duplicated using the COPY command. The command

COPY"FILE 1 .BAS" TO "FILE2.BAS"

makes the Basic file FILE2.BAS a copy of FILE1.BAS. Both files then exist on the disk. Note that it is essential to include the filetype (in this

case BAS) of both files. The new filetype need not be the same as the old one, but it is advisable to keep to a reasonably logical system.

If the purpose is just to give the file a new name (i.e. the original is no longer required) then the command to use is

RENAME"FILE1.BAS" TO "FILE2.BAS"

This has the same effect as the COPY command, except that FILE 1 is deleted.

The copy command is particularly useful when files are to be transferred from disk to disk (when more than one disk drive is available).

COPY "1 :PROG.BAS" TO "2 :PROG.BAS"

copies PROG.BAS from disk 1 to disk 2.

You have some choice in the way you refer to files. The drive number may come before or after the filename, provided there is a colon between the two. i.e. 1:PROG.BAS and PROG.BAS:1 refer to the same file. Also, you can replace the "." by *"/"*. i.e. PROG.BAS and PROG/BAS refer to the same file.

Housekeeping

If you continue to create new files for long enough, the disk will eventually fill up. This can be anticipated by keeping an eye on the number of free bytes (using either the FREE function or the DIR command). If you try to save a file which is too large, the file will be created, taking up all free space on the disk, but a DF ERROR will occur. The file created may not be complete and should be deleted and resaved on another disk.

To avoid this problem, it is good practice to delete all unwanted files as soon as they become unwanted. This may be achieved with the KILL command.

KILL"PROGRAM .BAS"

will delete the file PROGRAM.BAS. As with COPY and RENAME, the filetype (e.g. BAS) must be included.

Putting the Pieces Together

A Basic program held on disk may be combined with a Basic program held in memory using the MERGE command. The dominant program is the program on disk, so if there are any program line numbers in

common it will be the disk versions which are retained. The resulting program will reside in memory; the original disk version will remain on disk.

For example, if file FILE1 .BAS contains

```
10 INPUT"A";A
20 INPUT"B";B
30 SUM=A+B
40 PRINT "SUM=";SUM and FILE2.BAS contains
25 INPUT"C";C
30 SUM=A+B+C
```

then the effect of typing

```
LOAD"FILE 1"
MERGE"FILE2"
```

will be the following program in memory:

```
10 INPUT"A";A
20 INPUT"B";B
25 INPUT"C";C
30 SUM=A+B+C
40 PRINT"SUM= ";SUM
```

For Your Protection

To avoid the consternation of typing KILL"PROG.BAS" and then discovering you have deleted two hours work, it is worth learning about PROTECT.

Any file may be protected from over-writing and deletion using the command

```
PROTECT ON "FILE.BAS"
```

Commands such as KILL"FILE.BAS'~ will then fail, until the protection is removed by

```
PROTECT OFF "FILE.BAS"
```

While it is protected, a reverse screen "P" will appear against the filename in the directory of files.

A Spare Copy — Just in Case

For those who don't like all their eggs in one basket, there is a simple method of creating backup disks. If you have more than one drive, this is achieved "at a stroke".

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

will create a track by track backup copy from the disk on the *source* drive to the disk on the *dest* drive. The parameters *sides* and *tracks* are respectively the number of sides (1 or 2) and tracks (40 or 80). The default values are *source*=DEFD, *dest*=DEFD, *sides*=1, *tracks*=40.

Clearly, when *source* and *dest* are the same (as in the default values) there is a problem concerning two disks and only one drive. Under these circumstances, when the BACKUP command is used, your Dragon will instruct you alternately "INSERT SOURCE" and "INSERT DESTINATION" until the whole content has been transferred.

Directory tracks are always verified during BACKUP; other tracks are verified if VERIFY is "ON" (the normal power-up condition). This verification can be suppressed using the command VERIFY OFF, and re-activated with VERIFY ON.

As the BACKUP command uses whatever space is available, to obtain faster copying delete the contents of memory by typing NEW before starting a BACKUP.

Some Useful Functions

Information about files is given in the file directory, which can be called up using the DIR command. Lengths of individual files and the amount of free space on a disk may be found using functions FREE and LOF.

PRINT FREE *drive*

will give the number of bytes free on drive number *drive*. The default for *drive* is DEFD.

PRINT LOF"FILE.BAS"

will give the length of file FILE.BAS in bytes. The filetype (e.g. BAS) must be specified.

These may also be used in programs, e.g.

```
10 X=FREE
20 Y=LOF"PROG BAS"
30 PRINT"THERE ARE";X;"BYTES FREE,"
40 PRINT"AND PROG TAKES";Y;"BYTES."
```

CHAPTER3 READING AND WRITING

Apart from the obvious speed advantage that disks have over cassettes, we have seen that loading files is easier because of the directory of filenames – you don't have to read through a dozen files before you find the one you want (or, alternatively, you don't need a case full of cassettes!). This advantage is very important when it comes to reading and writing data.

Data may be stored on disk in two ways: in files, as programs are stored, or by writing directly to a particular sector on a particular track.

Data Files

The easiest way to use data files is via the simplest form of the FREAD and FWRITE commands. To use these simple forms of the commands you do not need to create the file in advance, or "OPEN" it.

The program

```
10 FWRITE"FILE";X,Y,Z
```

will create a file FILE.DAT (provided it doesn't already exist) and record in that file the values of X, Y and Z. Subsequently, the program

```
20 FREAD"FILE";A
```

will read in from file "FILE.DAT" the three values as one number. This is because the FWRITE command does not write a terminator after each value, so that the more extended form of the command (see page 14) can be used. To overcome this, the terminator must be forced in the following way

```
10 FWRITE"FILE";X,",",Y,",",Z
20 FREAD"FILE";A,B,C
```

One problem which arises with strings is demonstrated in the following program:

```
10 X$="THIS IS IN FACT, A SENTENCE."
20 FWRITE"OUTPUT"~X$
30 FREAD"OUTPUT";A$
40 PRINT A$
```

The result of running this program is

```
THIS IS
```

i.e. the comma has been taken as a terminator for the string.

The program may be continued

```
50 FREAD"OUTPUT";B$,C$
60 PRINTB$:PRINTC$
```

which gives

```
THIS IS
IN FACT
A SENTENCE
```

To overcome this problem we have the FLREAD command.

```
10 X$="THIS IS, IN FACT, A SENTENCE"
20 FWRITE"OUTPUT";X$
30 FLREAD"OUTPUT";A$
40 PRINT A$
```

The new program gives

```
THIS IS, IN FACT, A SENTENCE.
```

In the form described, the FWRITE command will always start writing at the present "end of file", extending the file as it goes. It is therefore advisable to delete previous versions of data files before re-running programs.

For more sophisticated use of data files, the commands FWRITE, FREAD and FLREAD may be used with parameters:

FWRITE"FILENAME", FROM *start,* FOR *length;* variables
FREAD"FILENAME", FROM *start,* FOR *length;* variables
FLREAD"FILENAME", FROM *start,* FOR *length;* string

The FROM parameter *(start)* selects the byte in the file at which the record starts. For FWRITE it defaults to the end of the file; for FREAD and FLREAD it defaults to the beginning of the file, when a "read" is first performed.

The FOR parameter *(length)* determines the length of the record to be written (in the case of FWRITE) or the number of bytes by which the byte count is advanced (in the case of FREAD and FLREAD). Its default value is equal to the length of the record being processed.

Before a FROM value can be used with FWRITE, the position in the file (and the file itself) must already exist. This may be achieved with the CREATE command.

CREATE"FILE",size

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

The following is an example of the use of these commands.

```
10 X$=”FIRST”:X=50:Y$”SECOND”:Y80
20 FWRITE”DATAFILE”,FOR20;X$
30 FWRITE”DATAFILE”,FOR20;Y$
40 FWRITE”DATAFILE”,FROMl0;X
50 FWRITE”DATAFILE”,FROM30;Y
```

Provided DATAFILE.DAT does not exist already, line 20 will create it, giving it length 20, with the record “FIRST” at the beginning. Line 30 writes “SECOND”, starting at the current end of file and extends the file by 20 bytes.
Line 40 inserts the value of X at byte 10.
Line 50 inserts the value of Y at byte 30.
Note that if line 40 read

```
40 FWRITE”DATAFILE”,FROM10,FOR20;X
```

then the record “SECOND” would be overwritten with blanks.

The following may be used to read the record

```
60 FREAD”DATAFILE”;A$,A,B$,B
70 PRINTA$,A,B$,B
```

This results in

```
FIRST          50
SECOND         80
```

Another way to read the values only is

```
60 FREAD”DATAFILE”,FROM1O,FOR2O;A
70 FREAD”DATAFILE”;B
80 PRINTA,B
```

Here, line 60 reads the value of A, starting at byte 10, and advances the read pointer to 30 (adding 20 to the start of 10) ready for the reading of B in line 70.

The function LOC may be used to determine the position of the read pointer in a file.

```
PRINT LOC”FILE”
```

gives the next byte to be read in FILE.DAT.

Simulated Random Access

The FROM/FOR form of address allows indexed records to be stored and retrieved in any order. 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"MAXIMUM LENGTH OF EACH RECORD";L
40 CREATE N$,N*L+20
50 FWRITE N$,FROM 0, N
60 FWRITE N$,FROM 10; L
70 INPUT"RECORD NUMBER";I
80 PRINT"RECORD";I;
90 INPUTR$
100 FWRITE N$, FROM(I~1)*L+20, FOR L; R$
110 GOTO7O

SAVE"STORE"

10 INPUT"FILENAME";N$
20 FREAD N$, FROM 0; N
30 FREAD N$, FROM 10; L
40 INPUT"RECORD NUMBER";I
50 FLREAD N$, FROM(I~1)*L+20, FOR L; R$
60 PRINT R$: GOTO4O


SAVE"RETRIEVE"
```

The data file is created using

```
RUN"STORE"
```

The program first asks for a filename (any legal filename may be given); then the number of records. As a test, give this the value 5. Maximum record length (the longest string you intend to store), say 20. Try entering the following records (in any order):

1 ALICE
2 BOBBY
3 CAROL
4 DAVID
5 EDWARD

Records may be overwritten, e.g. we could replace 4 DAVID with 4 DAWN. Then the program is terminated with the [BREAK] key. Now the records may be retrieved using

RUN"RETRIEVE"

The filename must be specified again, but the other parameters are read from the file. Now if we enter a number from 1 to 5, we retrieve the relevant record.

Too Many Files!

The FWRITE, FREAD and FLREAD commands may be used with different filenames simultaneously, but every time a new file is created or accessed it is left "open". Up to ten files may be open at any particular time. (A file may be opened for read and write simultaneously, but only counts as one file.) Any attempt to open an eleventh file will result in a TF ERROR. All files on a particular disk may be closed using the CLOSE command.

This has the effect of setting all read pointers (accessed by function LOC) to zero.

CLOSE *drive*

closes all files on the disk on drive number *drive*. If *drive* is not specified, all files on all disks are closed.

Is This The End?

A problem which frequently presents itself when reading records from a file is knowing when to stop. A useful function for this situation is EOF. If you include the statement

X = EOF("FILE")

in your program, X will usually take the value 1. But if the read pointer is at the end of FILE.DAT (i.e. there are no more records to read) it will have value 0.

A typical use of EOF is in loading an array from a file, when you are not sure how many values you have to read.

```
10 I=0
20 FREAD"DATAFILE";A(I):I=I+ 1
30 IF EOF("DATAFILE")=l THEN 20
```
Of course, you will usually need a DIM statement.

Reading and Writing Without Files

When a disk has been formatted, it is divided into tracks, and each track is divided into sectors. The standard drive works with 40 tracks, and 18 sectors per track. Each sector is 256 bytes.

Each 256-byte sector can be individually accessed by the SWRITE and SREAD commands.

To write a record to a sector, the record is assembled in two strings, e.g. X$ and Y$, each having a maximum of 128 bytes. The command to use is then

SWRITE *drive, track, sector,* X$, Y$

where *drive* is the disk drive number (1-4), *track* is the track number (0-39 or 0-79) and *sector* is the sector number (1-18 or 1-36). X$ and Y$ may be replaced by any strings, and may be ordinary strings containing letters, numbers and symbols, or they may be assembled using the CHR$ function. If either of the strings is less than 128 bytes, the spare space in the sector will be filled with CHR$(0) characters.

To retrieve data directly from a sector, the command is

SREAD *drive, track, sector, X$,* Y$

where all parameters are as in SWRITE.

Whether or not the strings written to the sector were of full 128 bytes length, the strings now retrieved will be. They will each contain the bytes of the appropriate original string, followed by a series of CHR$(0) characters. It will be necessary to CLEAR sufficient string space for the full 128 bytes each.

CHAPTER4 MORE POWER TO YOU DRAGON

In addition to the specifically disk-orientated commands, the DOS cartridge contains several new commands and functions to enhance your Basic interpreter.

Where Have I Put My Memory?

Two useful additions to the MEM function are HIMEM and FRE$.

PRINT HIMEM
gives the highest memory location available to Basic. On power-up this has value 32766. If space is reserved for machine code or data using the second parameter of the CLEAR command, then HIMEM will take the value 1 less than the second parameter. e.g. the program

```
10 CLEAR 200, 32000
20 PRINT "TOP OF BASIC MEMORY =",HIMEM
```

gives the result

```
TOP OF BASIC MEMORY = 31999
```

PRINT FRE$
gives the number of free bytes available for strings (at power-up it is 200). Although it ends with a "$", FRE$ is a numeric variable, not a string. This is particularly useful for programs which require strings to be entered from the keyboard. When string space is getting short, a suitable message can be displayed suggesting a "dump" to disk. A suitable program statement would be

```
IF FRE$<255 THEN PRINT "DUMP REQUIRED"
```

Reference to FRE$ causes a "garbage collection", i.e. the storage of strings is reorganised to make most efficient use of the space available. It, can therefore be used during intentional pauses in a program to avoid those embarrassing unplanned pauses that sometimes occur in large string-handling routines.

Don't Stop For Errors

Whenever your Dragon detects an error, it will stop the program and give you an error message. For most programming errors, it is essential to stop and correct the program. But some errors are caused by "operator error" such as keying in wrong data. The following program provides a simple example.

```
10 INPUT"X";X
20 INPUT"Y";Y
30 PRINT"X DIVIDED BY Y =",X/Y
40 GOTO10
```

The obvious way that an operator could "crash" this program is by giving Y the value zero. The program will then stop with a /0 ERROR IN 30 message. Of course, we can always test every value of Y as it is entered to check for zero, but a more convenient (and more powerful) method is now available.

The statement ERROR GOTOn tell 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 a code number (each type of error has a different code number).

The code for /0 is 20 so the following lines may be added to the program to deal with the error.

```
5 ERROR GOTO5O
50 IF ERR=20 THEN PRINT "CANNOT DIVIDE BY ZERO":GOTO10
60 PRINT"ERROR NUMBER";ERR;"IN LINE";ERL
```

Now if we input 15 for X and 0 for Y, we have the "CANNOT DIVIDE BY ZERO" message, but the program continues. If we give X the value 1E50 we have "ERROR NUMBER 10 IN LINE 10" (error 10 is OV ERROR).

There is a list of error codes at the back of this manual in Appendix 2.

On The Third Stroke. . .

A command that's guaranteed to attract attention is BEEP. Included in a Basic program, BEEP produces a decent sounding "beep". BEEPn produces *n* of them.

Hold On a Moment!

For those occasions when your screens of text disappear before people have had a chance to read them, we have WAIT. WAITn suspends execution of the program for *n* milliseconds. So WAIT1000 gives you a second, and WAIT10000 gives you 10 seconds etc. Here is a simple example.

```
10 CLS
20 PRINT "YOU SHOULD HAVE TIME","TO READ THIS."
30 WAIT2500
40 CLS
```

Fair Exchange

A useful function, especially in areas like "sorting" programs, is SWAP, which allows you to exchange the values of two variables, without the tedious problem of a temporary storage variable.

SWAP X,Y

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

Take a Hundred Lines

if you have ever typed in a line with the wrong number (e.g. 11 instead of 110) you will be aware that it can be quite a problem. Not only is the line missing from where it should be, it also turns up where it shouldn't be — and has probably over-written another important line. This sort of thing won't happen if you use AUTO, for automatic line numbering.

AUTO *start, increment*

will give you numbers starting with *start* and going up by the amount *increment*. You type your lines in, ending each line with ~NTER], then to return to normal mode just type ~NTER]. AUTO overwrites whatever is in memory on the particular lines being input, but does not "NEW" the program, so it can be used to insert lines. The default values for *start* and *increment* are respectively 100 and 10.

The following example shows how AUTO can be used.

```
AUTO100,100
OK
100 PRINT"FIRST PAGE"
200 PRINT"SECOND PAGE"
300 PRINT"THIRD PAGE"
400 [ENTER]
```

We may decide at this point that we should have used some CLS statements:

```
AUTO50, 100
OK
50 CLS
150 CLS
250 CLS
350 CLS
450 [ENTER]
```

But this is speed reading taken to extremes! So we must insert some pauses:

```
AUTO120,100
OK
120 WAIT2500
220 WAIT2500
320 WAIT2500
420 [ENTER]
```

This has the desired effect.

Booting Another DOS

For loading other operating systems from disk, there is a BOOT command. This loads the system into memory at byte 9728. The command to execute it is then EXEC9730.

The BOOT command takes the form

```
BOOTn
```

where *n* is the number of the disk drive (default DEFD).

APPENDIX 1

DRAGONDOS COMMANDS

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 FRE$ as the first statement of the program. For example:

```
10 ZZ = FRE$ ‘FORCE GARBAGE COLLECTION
20 REM REST OF CHAINed PROGRAM FOLLOWS
```

CLOSE

The command CLOSE is used to close files which have been opened by commands such as FWRITE, 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 lsts 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 typic al directory of files is as follows:

```
DIR
BASPROG        .BAK  1855
MACHCODE       .BAK  3809
BASPROG        .BAS  1855
DATAFILE       .DAT  3000
MACHCODE       .BIN  1009
163072 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 10 ERROR (input/output)

```
PRINT ERR
```

will return the value 42 (the code for 10).

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 over-ride 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 60; string
reads a string from FILE.DAT, starting at byte 100. After reading, the read pointer is advanced to byte 160 (100+60).

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 th 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 FREAD 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
```

FRE$

The function FRE$ gives the number of free bytes available for strings. It is not a string: it has a numeric value. Use of FRE$ forces a “garbage collection”.

PRINT FRE$
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 32766, 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”BPROG”
will load the Basic program from file BPROG.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 number 3 and RUNs it.

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 over-written.

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

The function 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.

WAIT5000

pauses for 5000 milliseconds (5 seconds).

APPENDIX 2

ERROR CODES

0	NF	NEXT without FOR
2	SN	Syntax Error
4	RG	RETURN without GOTO
6	OD	Out of data in READ
8	FC	Illegal function call
10	OV	Overflow
12	OM	Out of memory
14	UL	Undefined line
16	BS	Bad subscript
18	DD	Redimension Array
20	/0	Division by zero
22	ID	Illegal direct statement
24	TM	Type mismatch
26	OS	Out of string space
28	LS	String too long
30	ST	String too complex
32	CN	Can’t continue
34	UF	
36	FD	Faulty data
38	AO	File already open
40	DN	Drive number
42	I0	Input/Output Error
44	FM	Wrong File Mode
46	NO	File not open
48	IE	Input past EOF
50	DS	Direct statement
128	NR	Not Ready
130	SK	Seek
132	WP	Write Protect
134	RT	Record Type
136	RF	Record Not Found
138	CC	Cyclic Redundancy
140	LD	Lost Data
142	BT	Boot
144	IV	Invalid Directory
146	FD	Directory Full
148	DF	Disk Full
150	FS	File Spec.
152	PT	Protection

154	FE	Read past EOF
156	FF	File Not Found
158	FE	File Exists
160	NE	Non-existent
162	TF	Too Many
164	PR	Parameter Open
166	??	

	Page No.
AUTO command	21,24
BACKUP command	11,12,25
Backup files	*9*
BAK filetype	6,9
BAS filetype	6,7
BEEP command	20,26
BIN filetype	6,9
Binary files	9,46,53
BOOT command	22,27
Cassette programs	5
CHAIN command	7,8,28
CLOSE command	17,29
COPY command	9,10,30
CREATE command	14,3 1
DAT filetype	6,13
Data files	13
DEFD (definition)	5
Deleting files	10,45
DIR command	6,32
Directory of files	6,32
Disk, Backup	11,12,25
Disk, Formatting	5,34
Disk, Free spac~	6,12,32,41
Disk, Full	10
Disk, Initialising	5,34
Disk, Inserting	4
Disk, Type	4
DOS, Booting	22,27
DOS, Connecting up	4
DRIVE command	5,33
Drive number (in filename)	10
Drive number, selecting	5,33
DSKINIT command	5,34
End of file	17,18,35
EOF function	17,18,35
ERL function	20,36
ERR function	20,37
Error Codes	19,20,37,60

ERROR GOTO command 20,38
Exchanging variable values 21,55
EXEC default address 8,9,46,53
File-length 7,12,32,48
Filenames 6
Files, backup 9
Files, binary 9,53
Files, copying 9,30
Files, data 13,14
Files, deleting 10,45
Files, equivalent names 9,10
Files, incomplete 10
Files, list 6,32
Files, merging 11,49
Files, open 17,29
Files, protecting 11,50
Files, renaming 9,51
Files, transferring 10,30,51
Filetypes 6
FLREAD command 14,16,17,39
FOR parameter 14,15,16,39,40,43
Formatting Disks 5,34
FRE$ function 19,42
FREAD command 13,14,15,16,17,18,40
FREE function 12,41
FROM parameter 14,15,16,39,40,43
FWRITE command 13,14,15,16,17,43
Garbage collection 19,42
HIMEM function 19,44
Indexed records 16
Initialising disks 5,34
KILL command 10,45
Line numbering, automatic 21,24
LOAD command 7,46
LOC function 15,47
LOF function 12,48
Machine code files 8,9,46,53
MERGE command 10,11,49
PROTECT command 11,50
Protection 11,50
Random Access 16

Read pointer	15,17,29,35,39,40,47
RENAME command	10,51
RUN command	7,8,52
SAVE command	6,9,53
Sectors	18
[SHIFT] [@]	6
SREAD command	18,54
String space, free	18,19,42
SWAP function	21,55
SWRITE command	18,56
Tracks	18
VERIFY command	12,57
Video RAM	5
WAIT command	20,21,58

Printed by Cleglen Publishing Ltd., Cardiff

Cleglen 83—C4507

Dragon Data Ltd.,
Kenfig Industrial Estate,
Margam,
Port Talbot,
West Glamorgan.
SA13 2PE.

WWW.DRAGONDATA.CO.UK