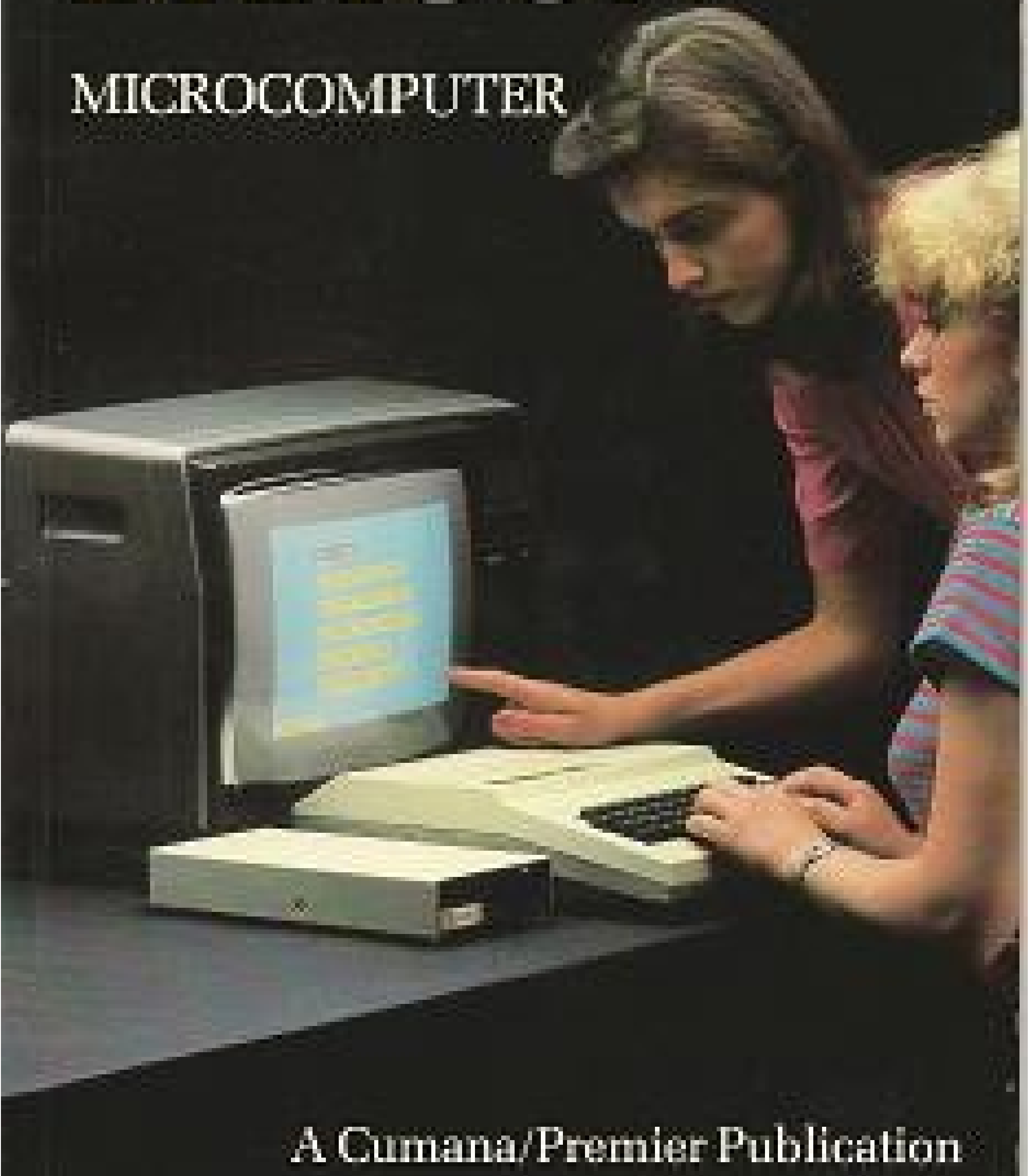
Using Floppy Disks with the DRAGON MICROCOMPUTER



A Cumana/Premier Publication

Using Floppy Disks with the Dragon Microcomputer

Justin Johnson/Keith Davis

A Cumana/Premier Publication

Published by

Premier Microsystems Ltd,
208 Croydon Road,
London SE20 7YX.
England.

Cumana Ltd,
Unit 1, The Pines Trading Estate,
Broad Street, Guildford,
Surrey GU3 3BH, England.

British Library Cataloguing in Publication Data

Johnson, Justin
Using Floppy Disks with the Dragon microcomputer
1. Dragon (Computer) 2.Computer storage devices
I. Title II. Rihan, Peter III. Davis, Keith
001,64'42' QA76.8.DI

ISBN 0 9508762 2 4

Typeset and printed in Great Britain
at FD Graphics, Fleet, Hants.

Contents

Introduction to Data Storage

This book has been written with the newcomer in mind and is based around the DRAGON Microcomputer fitted with the DELTA Disk cartridge.

There are a number of different data storage devices that are needed for the efficient operation of all computers whether they be large scale main frames or micro computers. There is the ROM or Read Only Memory, this holds fixed data which cannot be altered. This data is usually the program which holds the BASIC interpreter. It also tells the microprocessor how to look after such things as keyboard entry and how to handle the data. Then there is the RAM or Random Access Memory, this usually holds the current running program and its variables. As the name implies, data can be stored or retrieved in a number of ways. Programs are normally loaded into the RAM area as they are needed. When a new program is required it is loaded into the same RAM area and overwrites the old stored program. It can only hold data whilst the computer is turned off then all data and programs in the RAM are lost and only that data which is fixed into the ROM is kept. There are other types of memory whereby the program data and files can be downloaded before turning the computer off. One of these is the punched tape on which most of the early computers used to store and load their programs. Punched tape again holds fixed information and it cannot practically be altered. Then there is magnetic tape. This type of storage media has the advantage that data can be stored onto magnetic tape and filed away. The data on the tape can also be changed when required to hold a new program. It has several advantages over the punched tape. With a Personal Computer this convenience extends to being able to use a normal cassette recorder for long term storage and retrieval of programs and data.

Faster Access

Whether you have just started in computing or have had the use of a Micro for some time, sooner or later you will come to feel that the use of a tape recorder to save and retrieve programs can become frustrating. Initially, this tape recorder was just fine and allowed you to save and retrieve your brand new programs. But now you have a larger collection of software and find yourself spending rather a long time searching for the latest version of your Home

Management package. This normally happens when your program library grows to an extent whereby finding a particular program halfway down a tape is a time consuming business. This seems especially true just as you want to show a friend your latest masterpiece of programming power.

It is at this point that you begin to wish you had bought a better storage and retrieval system. This is where the Floppy disk drive fits the bill so nicely. With data transfer rates a couple of hundred times faster than cassette. Improved reliability in loading and saving programs and a “Directory” of diskette contents available to you in a few seconds rather than several minutes.

With the cost of Floppy disk drives now within the reach of most Micro owners and the promise of lower prices to come, it seems only right that we try to de-mystify the operation and facilities offered by a typical disk based system and its associated D.D.S. (Short for DELTA Disk System).

The Floppy Drive

Figure 1 shows a typical Floppy disk drive and the following points should be noted:-

{a} Indicator lamp illuminated to show that the drive is active and busy.

{b} Drive door and slot for inserting floppy diskette. This door is normally closed after inserting the diskette. It engages the hub or clutch of the drive into the hole of the floppy diskette. Note the way the diskette is fitted into the drive. The diskette label is uppermost and on the rear edge. The small notch on the left is the write protect notch and if covered by a small adhesive tab will prevent the drive electronics from storing information onto the diskette. This is useful if you wish to avoid accidental erasure of certain diskette data.

See Figure 2 for explanation of the floppy diskette.

{c} Control cable. This carries the signals needed to operate the drive. It is through this cable that all communication between the disk drive and the computer are carried out. It must be inserted the correct way round. If the cable is reversed then normally this will result in the drive motor and electronics staying permanently in the “ON” condition. If a diskette is in the drive at that time then one or more of the tracks will be totally erased.

{d} ON/OFF switch. It is normal practice to power all external devices prior to applying power to the computer. Always wait till the entire system is powered before inserting a diskette into the drive and be sure to remove it before switching anything in the system OFF.

Standards

There are two normal standards for 51/4" disk drive capacity. These are 40 track and 80 track drives. The only difference between 40 and 80 track drives of the same make are internal and transparent to the user. The 40 track drive lays down its magnetic tracks at a pitch of 48 tracks per inch or as is commonly known 48 TPI, whilst the 80 track drives lay down the tracks at 96 TPI, exactly half the track width of the 40 track drive.

The Diskette

When a diskette is first used it has to be Formatted or Initialised. This is a program that lays down magnetic circular tracks onto the

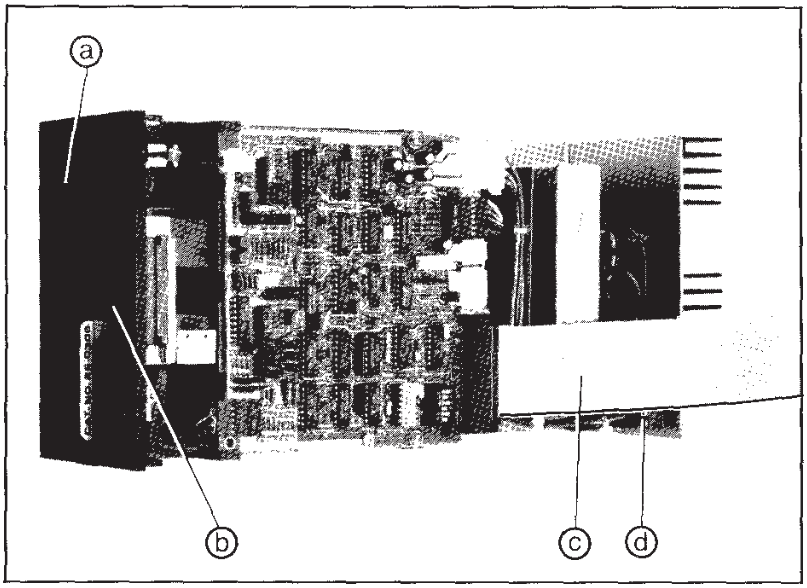


Figure 1. Floppy Disk Drive.

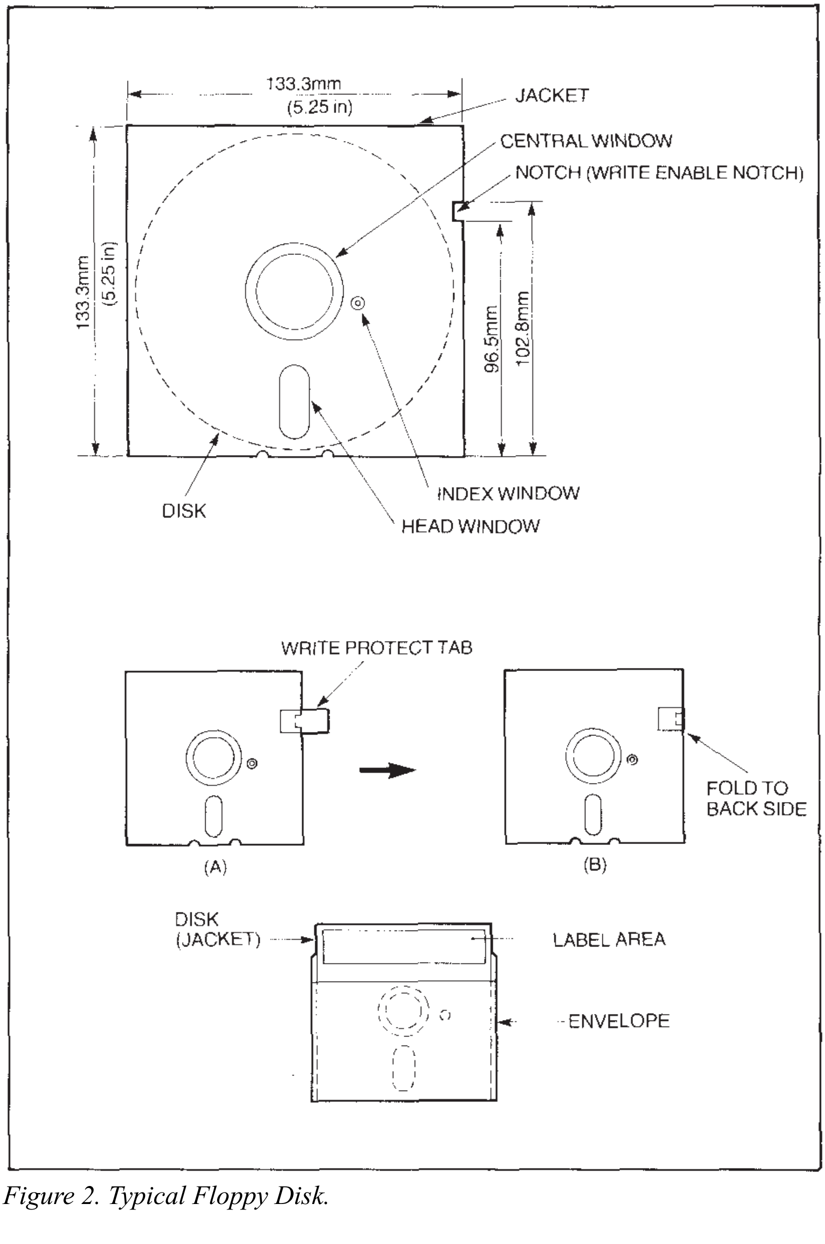


Figure 2. Typical Floppy Disk.

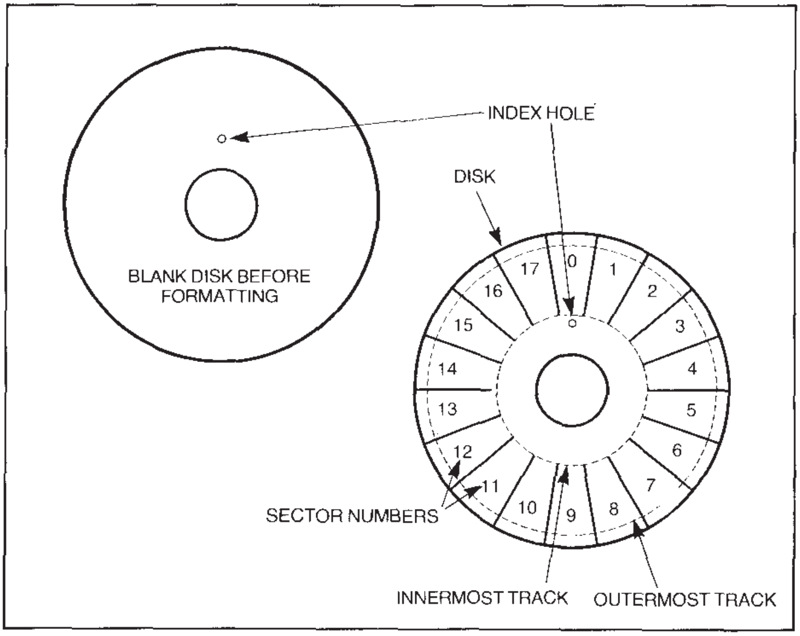


Figure 3. How the disk surface is laid out after format.

diskette and then segments each circle into a number of sectors. Each one of these sectors has space to store 256 bytes of information. In the case of the Dragon Micro fitted with the DELTA Disk cartridge, which normally operates in the DOUBLE DENSITY mode, it lays down 18 sectors per track. See Figure 3. That is equal to a storage area of 4,608 bytes on each track. Therefore a 40 track drive will have a formatted capacity of 40 times 4,608 or 184,320 byte of storage area. As I track of the Diskettes is reserved for the Directory, a total of 179,712 bytes are available for program storage. The storage area available on an 80 track drive, less the Directory, is equal to 359,424 bytes.

For the sake of convenience the drive storage capacity for the Dragon Micro is normally given to the nearest 100k of a formatted disk. Therefore a 40 track single sided drive is quoted as being 200k capacity whilst the 80 track single sided drive is quoted as being 400k and finally the 80 track double sided is known as a 800k drive. These figures are assuming single units. If the drives are in a dual packaged unit then the above figures are doubled.

Density and Capacity

Note that the previous calculations are made for single sided DOUBLE DENSITY 40 or 80 track drives. The 80 track drive is simply a higher capacity drive. The recording density remains the same. Even if the user goes to double sided drives, the Dragon Micro normally does its reading and writing in DOUBLE DENSITY. With a SINGLE DENSITY system the computer usually uses only 10 sectors per track. To imply double density means that the number of sectors per track has been increased from say 10 sectors per track to 18 or 20 sectors per track. The DELTA disk cartridge has been designed to operate at 18 sectors per track. It is a common mistake to think that double density implies double the amount of tracks. It is the host computer that decides which density it will operate in. Most of today's modern floppy disk drives will handle either density and no setting has to be done by the user. It is a function completely controlled inside the computer.

Directory Track

As mentioned above, one of the tracks is used for the directory. This directory is used for the management and location of the programs that are stored on the diskette. When a program, or file as it is more commonly called, is saved on a floppy drive then an automatic directory entry is made along with the file name that was given to it. This entry contains the track number and the sector where the computer can find the start of the file. Secondly it contains the number of sectors that the computer has to read into memory when it loads the program. The DELTA Disk System looks after the Disk drive and is able to find all spare sectors on a diskette when it is required to store a file. It does this by moving the READ/WRITE head via a stepping motor inwards and outwards across the rotating diskette. On a single sided disk drive the Read/Write head normally operates on the underside of the diskette. Whilst on a double sided drive both sides of the diskette are used. No user intervention is required when saving or loading files from a floppy disk drive. Certainly there is no rewind or setting to record as would be found on a cassette tape storage system.

Ground Rules

The type of floppy diskette used on the Dragon Micro is more commonly known as a mini-floppy. This is a 5¼" diskette made from a mylar coated circular disk enclosed in a stiff outer sleeve. The mylar coated diskette rotates inside the sleeve at about 300 RPM.

The word floppy is usually reserved for its larger brother the 8" disk.

Each box of diskettes normally comes with a set of diagrammatic instructions of what to do and what not to do. Here are a few Do's and Dont's. Never write on the diskette sleeve or label attached to the sleeve. Always mark up the label BEFORE placing it on the diskette sleeve. Never place the diskette within any magnetic field like a loudspeaker, mains transformer, moving coil meter etc or anything that you suspect to be magnetic or that could otherwise damage your diskettes.

When the diskette is not in use or in the drive then return it to its protective jacket. Never. leave an unprotected diskette on any surface where It might pick up damaging particles of dust or grit. Grit in particular can cause serious damage to the precision Read/Write head used in the drive. Avoid excessive bending of the diskettes and if possible keep them in the rigid plastic diskette holders that can be obtained from most computer shops. Do not touch the exposed recording surface of the diskette. Finally do take care to keep your diskettes away from excessive heat or cold

In selecting the right type of diskette to use always choose a branded type of diskette that is certified for the number of tracks that you wish to use them on. Users of 80 track drives can expect to pay slightly more for certified 77/80 track diskettes. If you change from 40 to 80 track drives at any time then do not expect all your 40 track diskettes to reformat in the 80 track drives.

If one of your diskettes gets damaged in any way either physically or magnetically then try to copy the files of that diskette one at a time onto a known good formatted diskette. You should at least save most of your files and only those in the damaged area will be lost. If you have any really valuable software then it is up to you to keep backup copies of those diskettes in a safe place for such an eventuality.

Instructions on Initialising, Copying and Backup are given elsewhere in this book.

Setting Up

Now that we have had a brief look at the disk drive it is time to connect it to the host computer. In this case it is the DRAGON 32/64 computer fitted with the DELTA Disk System. Firstly, check that your drive or drives are addressed correctly as drive 0 and drive 1. This addressing can be checked in the case of Cumana drives by removing the drive cover and looking at the DIL switch that is located towards the rear right hand edge of the drive electronics PCB. The following switches should be forward (ON) for drive 0. This is normally called the boot drive. Switches marked DS0, HS and MX. On a dual unit the second drive will have switches DS1, HS and MX forward. All other switches should be made towards the rear of the drive or (OFF). In this configuration only the head of the selected drive will load when disk access is required. If it is desired to have all the heads loaded when the motors start up then the entire switch block should be removed and refitted one space to the left, on each drive. The following switch setting will then apply. Drive 0 will have DS0, HM and MX forward. Drive 1 on a dual drive system will have DS1, HM and MX forward. It is fairly easy to see if the settings have been made correctly. On a single drive system there will be two switches forward and four switches backwards whilst in a dual drive system there will be three switches forward and three switches backwards on each drive. See Figure 4 for Cumana switch settings.

Other Types

Other types of drives use a similar convention and the guidelines given for the Cumana drives should help you get the correct settings even if the drive uses links that have to be cut. Don't forget

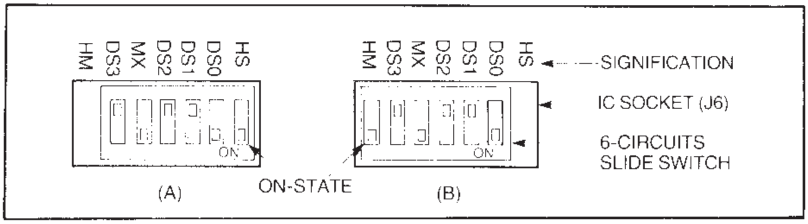


Figure 4. Mounting Position of the Switch.

that if the links are mounted in a DIL socket than it is quite easy to change the links for a DIL switch.

The Termination

A final drive check should be made on the termination resistor pack. This looks something like a small integrated circuit and in the case of the Cumana drives is normally white in colour and is mounted in another DIL socket. It has the word BECKMAN written on it. There should only one terminating resistor fitted to the drive assembly whether you have one or two drives. If you have a dual drive assembly and each drive has one of these resistor packs then remove the resistor pack from the drive that will be closest to the computer leaving the resistor pack in the drive that will be on the end of the cable.

The Drive Cable

Now fit the control cable to the drive. It is a 34 way ribbon cable with three connectors on it. Two of these connectors are close to each other at one end and are of the edge connector type. Fit the end edge connector onto the drive with cable coming away from the bottom of the connector. In the case of a single drive it is just possible to fold the cable so that the other connector is held within the case. With the dual drive ensuring that the orientation of the cable is kept the same. A red line runs alongside one edge of the cable to assist in this. There is a groove at the back of the drive housing where the cable may rest in as it exits out. When the lid is refitted to the assembly you will find that the cable is not crushed by the lid.

Plug Into The Micro

Now fit the other end of the cable onto the exposed section of the DELTA Disk cartridge. The cable plug is fitted to the edge connector with the cable going downwards and away from the pack. This is a 34 way connector and care must be taken to ensure that no force is used to fit it. It is possible to mis-align the connector and damage the printed circuit of the DELTA cartridge. Make sure there are no bent pins on the plug and that the edge connector is clean. Having successfully connected the drive to the Micro it is now time to test it out.

Introduction to Delta

The DELTA Disk System has been specially written by Premier Microsystems for use with the Dragon Microcomputer.

DELTA is suitable for all current types of popular floppy disk units, i.e 5.25 inch. A Dragon equipped with DELTA can cope with any of these units or even a mixture of different drive types. The Premier DELTA cartridge contains an advanced disk controller circuit and an 8K ROM containing the DELTA Operating System. The use of a ROM means that very little of the Dragon's valuable RAM is needed by the disk system. DELTA uses only 1.8K of user RAM, unlike some other disk operating systems which require 10 - 20K!!

DELTA has been written to be fully integrated with the Dragon's excellent BASIC. All existing BASIC words continue to work in the normal way. DELTA adds over 30 new or extended words to the BASIC vocabulary. These new words are used just like any other BASIC commands. In addition to the new facilities and commands DELTA provides comprehensive Disk Error messages. These are given in plain English making debugging easier.

The most advanced feature of DELTA is the data file handling. DELTA can handle up to 8 files on up to 4 separate drives simultaneously. These files can be Serial or Random Access (with variable record length). The file handling is done using simple BASIC commands.

This manual is designed to help you make the best use of the disk unit supporting your computer. If you already have a good idea about disks and backing stores, please skip ahead a few paragraphs. If not, please read this section carefully, plus the two sections entitled POINTS TO NOTE and GETTING STARTED before experimenting.

Disks are used for the long term storage of programs and data. This type of storage is known as a 'backing store' in computer parlance. The computer's internal memory or RAM (Random Access Memory) has two major limitations - it loses all its information once the computer is switched off and even the largest RAM could not hold simultaneously all the programs you will ever want to use!

The function of the RAM is to hold the program currently in use, while the backing store saves all other programs and program data. We can also use our backing store to pass information between programs by using data files.

Backing stores come in a bewildering number of forms: tape cassette, cards, paper tape, bubble memory, magnetic floppy tape, and various types of magnetic disk. Disk systems, although not the cheapest form of storage, are extremely fast in operation and easy to use.

The outer cardboard cover of a disk protects the inner revolving disk. The inner disk is made of mylar and coated with a very fine high quality magnetic material, similar to a conventional tape. NEVER touch the surface of the inner disk. The disk coating stores information as magnetic impulses which can be read by the disk drive. This type of storage is very reliable, provided the disks are properly stored and treated. See Points To Note for what you should and shouldn't do with disks - especially if this is your first experience of disk operation.

The microdisk contains a similar recording media, but is enclosed in a strong plastic case. A safety catch ensures that the internal disk cannot be handled!

The drive lays down information on concentric rings on the surface of the disk. The outermost ring is known as track zero and the tracks are then numbered inwards towards the centre of the disk. The actual number of tracks used varies from drive to drive. Each track is subdivided into sectors, the quantity again depending on the system in use.

The computer itself controls the disk unit using a special internal program known as the DOS (Disk Operating System). This program is supplied in your system cartridge. DELTA allows the user to give simple, easy to remember instructions to make full use of the disk. The DOS automatically finds programs, information or free space on the disk - the user does not need to worry about tracks or sectors.

Whenever a program or data information is put onto a disk, it is given a filename. The DOS maintains a directory on each disk which remembers which tracks hold which program. The 'DIR' command will display this disk directory onto the screen.

The DOS also gives the user a wide range of utility or 'housekeeping' commands which allow files to be deleted, re-named etc, should the need arise. The most important feature of DELTA is its ability to manipulate up to eight different data files on up to four drive units simultaneously. These files can be serial or random access or indexed.

DELTA supports six types of fi1e:-

1. Program
2. Serial
3. Random
4. Indexed
5. Machine Code
6. Executive

Detailed instructions for their use can be found in later chapters, but a short description of each below, explains their general purpose.

A program file is used to store programs written in a high level language such as BASIC or Assembler Source Code. Once a program has been written, it can be stored on the disk using the SAVE command. The same program can be recovered for use at a later date using the LOAD, RUN, APPEND or CHAIN commands. Programs can be APPENDed, renamed or even deleted using the various
DELTA utility commands.

A serial or sequential access file is used to hold data, either figures or text or both. In this type of file, information is recorded and recovered in a sequence (hence sequential file). To find one item of data it is usual to read the file from the start until the required item has been located. This type of system is very easy to program and makes efficient use of disk storage, but it does have its disadvantages as will be discovered later.

Random Access files reserve a space on the disk for each item or group of items of data. These spaces are known as 'records'. This system of filing allows particular items of data to be picked up directly from the disk, without reading any previous data, allowing much faster recovery of information from the disk. Unlike many other DOS systems, DELTA allows the user to change the length of these records using a simple DIM statement.

Indexed files are a combination of serial and random access files. A short serial or index file is used to locate larger blocks of information on big random access files. This is a particularly useful technique when large files are being used. The system is similar in concept to a card index in a library, but hundreds of times faster and more accurate!

For many applications, programs written in Machine Code are more suitable than BASIC. DELTA itself is an example of a large machine code program. The LOADM and SAVEM commands allow machine code programs to be moved into and out of the computer memory. These programs are stored in machine code files.

The Executive file is used to produce a logical order of executing programs once the relevant disk has been 'booted'. The BUILD ... DO sequence is used to achieve this.

At first sight, this may all seem rather confusing, but anyone who has already learnt BASIC should have little trouble with the new instructions needed to use DELTA. The BASIC we use is similar to that used in many other popular machines, although the file handling commands vary from machine to machine, with no set 'standard' file handling words.

We would advise you to work through the relevant chapters and examples in this manual. You will soon be able to write your own file programs. Experiment with your system but one strong word of warning!

DO NOT EXPERIMENT WITH YOUR MASTER DEMONSTRATION DISK

Make copies (see BACKUP) and experiment with them! If this is your first experience of disk operations, read POINTS TO NOTE and GETTING STARTED before experimenting.

Getting Started

Please read this section carefully before using your DELTA system. Do not make any connections to either the Dragon or the disk drive with the Dragon or drive turned on. Failure to observe these precautions may cause damage to the entire system.

Connections

Connect the DELTA cartridge to your disk drive(s) using a 34 way ribbon cable. This may necessitate removing the drive cover, depending on the design of drive in use. Make sure that the drive cover(s) is/are replaced properly on the drives before powering on.

Now carefully insert the cartridge into the DRAGON's cartridge slot, situated on the right hand side of the machine. Open the door of the disk drive but DO NOT insert a diskette yet.

Connect the drive and your DRAGON to the mains. Turn on the drive, then the DRAGON, in that order. The screen should clear and display the start-up message:-

```
DELTA SYSTEM FOR DRAGON 32
BY J G JOHNSON AND P J RIHAN
<C> 1983 PREMIER MICROSYSTEMS
PUT DISK IN 'A' <PRESS KEY>
```

The displayed message may vary slightly from the one above.

If the message does not appear, turn off and check all connections carefully, especially the orientation of connecting cables (see section on PROBLEMS). Do NOT insert a diskette into a drive until you have succeeded in displaying the start-up message.

Taking great care not to touch the brown surface of the diskette, insert the demonstration diskette into the drive with the title label facing the direction of the closing drive door and the oblong slot in the diskette nearest the drive (see diagram). Now gently close the door.

N.B. Never switch the disk drive or the Dragon on or off with a diskette resident - danger of losing entire diskette contents!

Next press any key. The drive will start up and automatically run the INTRO program. You have now 'booted' the disk!

Your DELTA system is now ready for use. Overleaf are some example operations to ease you into the DELTA Disk System as painlessly as possible! Note that the '<ENTER>' symbol used in this manual refers to the need to hit the ENTER key.

Summary

1. Connect cables, plug in DELTA cartridge.
2. Switch on drive(s) then DRAGON.
3. Boot disk.
4. Always INITialise a new disk before use. (DO NOT INITialise your demo disk!)

Down to Work!

The DRAGON will now obey all the new disk commands in addition to all the BASIC words you have been using up to now. Type:-

```
DIR <ENTER>
```

and assuming there is a DELTA demonstration diskette in the drive and the drive door is closed you should see a display of the diskette contents showing filenames, file types and lengths of the various demonstration files supplied.

To run the introduction program type:-

```
RUN "INTRO" <ENTER>
```

The program 'INTRO' will now load and start to run, displaying a welcome message followed by a DELTA system logo and signal. After a few seconds a menu will appear:-

```
1. UNLOCK SYSTEM
2. DISK DIRECTORY
3. DISK DICTIONARY
4. DISK PURGE
5. DISPLAY DELTA LOGO.
CHOICE?
```

An explanation of each choice.

1. NEWs the workspace and clears the screen ready for programming.
2. Prints a directory of the demonstration diskette.
3. Enter one of the new or changed BASIC words which DELTA gives you and a brief resume of its function is displayed.
4. Can be used to delete files from the diskette - use with care and don't use at all if you are unsure!
5. Displays the DELTA logo.

Try selecting numbers 2, 3 or 5 from the above menu. All will return you to the above menu after execution. When you have had enough select 1 and continue below.

Two file-handling programs have been included on the diskette. PINPUT is used to create and add names to a file of telephone numbers.
NUMBER is used to search the file built by PINPUT.
Type:-

```
LOAD "PINPUT" <ENTER>
```

to LOAD 'PINPUT' into the workspace.

This program uses Random Access files to hold data on names and telephone numbers. Each time the program is run the file is extended, adding new names to the existing list. If you ever wish to erase the file and start again type:-

```
FLUSH"PDATA"
```

PDATA is the name of the file which holds all the actual names and

numbers - PINPUT is merely the creation program, it does NOT store the information to disk under that filename.
Type:-
```
'RUN <ENTER>'
```

and add some names and numbers to the short list supplied. You may also like to LIST the program to see how it works – copious REM statements have been included to assist analysis.
Next type:-
```
LOAD"NUMBER" <ENTER>
```

NUMBER searches the file (PDAT A) for a name or number. If any part of the name or number is given, NUMBER can find the name and number that corresponds to the information requested. If you type:-
```
SMITH <ENTER>
```

the program will list all occurrences of SMITH, whether plain SMITH, SMITHE or ARROWSMITH. If you enter 698 it will list every phone number containing those consecutive numbers. Try entering SON to find how many of the names contain it – eg. Jackson, Johnson, Attsonton.

The program can even deal with uncertain spellings – you simply type an asterisk ‚*' for every uncertain entry. For example:-
```
J***SON <ENTER>
```

would find JOHNSON or JACKS ON as the search will ignore the starred portions.

LIST to see how the program works - you will find it is surprisingly short due to DELTA's powerful file-search commands. REM statements explain in detail what is going on.

These demonstration programs, plus the numerous examples will give you some ideas on the possibilities of disk data files. Start learning DELTA by typing in some of the examples in this manual, then modify them to your own requirements. File handling is extremely easy using DELTA, but you will need some practice, especially if your BASIC knowledge is weak. Don't be disheartened by silly mistakes.

REMEMBER
> > > IF ALL ELSE FAILS, READ THE MANUAL!! < < <

Note!!
All fresh disks MUST be initialised before DELTA can use them. See the INIT command summary for how to format a disk. Before
trying out some of the examples in this manual, you will need a small supply of disks.

Delta Keywords

Overleaf begins an explanation of DELTA's new command words. They have been arranged into logical groups, as follows:-

BASIC WORDS

SAVE	LOADM	CHAIN
SAVEM	RUN	APPEND
LOAD	RUNM	

DISK WORDS

DIR	VERIFY	FLUSH
INIT	SELECT	OPEN
CONFIG	COPY	CLOSE
KILL	BACK UP	FILES
ASSIGN	CREATE	

FILE WORDS

END#	IF EOF (N) THEN	PRINT#
RESTORE#	(LINE)INPUT#	FIND#
DIM#		

EXECUTIVE WORDS

BUILD	DO	BOOT

SAVE SAVE

Function
BASIC programs are saved on to a disk file using this command. If the named file does not already exist a suitable file will be created by the system. If a file already exists, the program will overwrite the original; this will not happen if the existing file is protected.

The file name may he up to eight characters long and should start with a capital letter (See FILENAMES). In a multi-drive system the file name may he prefixed with a drive specification. If this is omitted the system uses the currently selected drive.

An error message is generated if there is no disk space or if the existing file is of the incorrect type.

Syntax

SAVE "NAME"	
SAVE "B:NAME"	- on a multi-drive system B: = drive specification.

Examples

```
SAVE "EXAMPLE"
SAVE "A:EXAMPLE"
```

Comments

Used for BASIC programs - see SAVEM for machine code. LOAD recovers the program. For output to cassette use the CSAVE command syntax as normal.

Associated Keywords

LOAD	CHAIN	APPEND
RUN	CLOAD	CSAVE

SAVEM SAVEM

Function

Machine code and data in memory are saved using SAVEM. In the case of SA VEM the area of memory must he specified. The rules concerning file names and overwriting of files are the same as SAVE.

Syntax

SAVEM”NAME”,A1,Z1	A1 = start address, Z1 = end address.

Examples

10 SAVEM”SCREEN”,&H0400,&H05FF	saves text screen to disk
20 CLS	
30 LOADM”SCREEN”	restores text screen
SAVEM”B:BASIC.ROM”,&H8000,&HBFFF	transfers BASIC interpreter to disk

Comments
Start and end addresses may he specified in decimal or hexadecimal notation. It is usually easier to use Hexadecimal addresses. i.e. &H---- in the SAVEM command. In addition to saving M/C this command can be used to store graphic screens for later use. LOADM retrieves the SAVEMed code. Use SAVE for BASIC programs.

Associated Keywords

LOADM	RUNM	EXEC
CLOADM	CSAVEM	

LOAD LOAD

Function
To LOAD a BASIC program from disk to memory. The file name may contain an optional drive specification if needed in a multi-drive system.

Syntax
LOAD "NAME"
LOAD "C:NAME"

Examples
```
LOAD "INTRO":RUN
LOAD "A:NUMBER"
```

Comments
Only used for BASIC programs, see LOADM for M/C. Loading BASIC programs from disk into the computer automatically erases the resident program. An error message will he generated if the file does not exist or if the file is not a BASIC program.

To transfer a program from cassette to disk storage, simply CLOAD from cassette then SAVE"xx" to disk. You will find a separate chapter on cassette to disk transfer later in this manual.

Associated Keywords

SAVE	CHAIN	APPEND
RUN	CLOAD	CSAVE

LOADM LOADM

Function

Recovers M/C and data from disk and returns it to the memory.

Syntax

LOADM "SCREEN"	
LOADM "C:CODE",ADDR	optional LOAD address
LOADM "MCODE",&H5000	loads code to $5000 regardless of original location of code.

Examples

```
10 SAVEM "SCREEN",&H0400,&H05FF
20 CLS
30 LOADM"SCREEN"
40 LOADM"SCREEN" ,&H0500
```

Comments

Can be used to move machine code into different areas of memory. Unless otherwise specified the file is returned to its original position in memory. If the data is required in another position this is specified after the file name.

Error messages are generated if the file does not exist or if the wrong type. Drive specifications may be included in the file name. LOAD is used for BASIC programs.

Associated Keywords

SAVEM RUNM EXEC CLOAD CSAVEM

RUN RUN

Function

Loads then runs a BASIC program; see LOAD for details of loading. The program may be started at a specified line number, but NOT from a program line - immediate mode only.

Syntax

RUN"NAME"	
RUN"D:NAME"	
RUN"NAME"100	runs NAME for line 100

Examples

```
RUN"INTRO"
```

```
100 IF F$ ="ACCOUNT" THEN RUN "ACCOUNT"
110 IF F$ ="STOCK" THEN RUN "STOCK"
120 IF F$ ="PAY" THEN RUN "PAY": ELSE PRINT "ERROR":
      RUN
```

Comments
A useful command when using a suite of related programs; functionally the same as 'LOAD "NAME": RUN'. Can be used in the immediate mode to LOAD and RUN a program from disk or within programs to call in different programs; this is useful in menu-driven programs. The RUN from a line number facility can only be used directly from the keyboard, NOT from program.

Associated Keywords

LOAD	APPEND	CHAIN	SAVE	CSAVE	CLOAD

RUNM RUNM

Function
Loads and executes a machine code file. An optional start address may be specified. The code is executed from its beginning (see Comments).

For file name rules and errors - see SAVEM and LOADM.

Syntax
RUNM"CODE"
RUNM"C:CODE"

Examples

RUNM"ENCOD9"	
RUNM"MENU2",ADDR	optional LOAD address

Comments
Execution of the loaded file will ALWAYS commence at the start of the program - put a JMP at the beginning of your program if you wish to start elsewhere. Routines should be modified to start from the lowest address saved. If this is not convenient use the combination LOADM" NAME": EXEC&H----.

Associated Keywords

LOADM	SAVEM	EXEC	CSAVEM	CLOADM

Function

The chain command allows the variables in one program to be carried over to the next program. Normally when a fresh program is loaded or run the variables from the previous program are lost. A CHAINed program loads and runs but retains all the variables (numeric, string and array) from the previous program. This permits the use of programs that are too big for the computer's memory. The large program can be sub-divided into smaller units and variables passed between programs using CHAIN instead of RUN.

A small modification is needed to make sure that string variables are passed correctly.

When defining a string in a LET (or implied LET) the following syntax should be used:

A$=" "+"ABC" instead of A$="ABC"
or D$(K)=" "+"FRED" instead of D$(K)="FRED"
or LET K$=" "+"CHAIN" instead of K$="CHAIN"

Note that user defined functions cannot be carried over between chained program. This is necessary due to the way in which the BASIC interpreter stores strings in memory. No other changes are required in the programs. No modifications are needed unless the strings are to be used by a succeeding program.

Syntax

CHAIN"SECOND"
CHAIN"B:PART2"

Examples

```
990 IF A$="A" THEN A$=" "+ACCOUNT
1000 IF A$="ACCOUNT" THEN CHAIN "ACCOUNT"
1010 IF A$="PAY" THEN CHAIN "CASH"
1020 RUN"MENU"
```

Comments

Allows very long programs to operate in a limited amount of memory. Remember to modify all string definition statements and avoid user defined functions if required (see above).

Associated Keywords

RUN LOAD SAVE APPEND CLOAD CSAVE

APPEND APPEND

Function

Merges the program in the workspace with a program on the disk. Care must be taken with line numbers. RENUMBER should be used to prepare programs before APPENDing. In the event of two identical line numbers the incoming disk program takes precedence. More than one program may be appended. The same rules regarding names etc. apply as in LOAD.

Syntax

APPEND"NAME"
APPEND"C:NAME"

Examples

As syntax

Comments

Watch out for duplicated line numbers. APPEND is a slower function than LOAD so a long disk program may take some time to merge.

Associated Keywords

RUN CHAIN LOAD SAVE CSAVE CLOAD

DIR DIR

Function

Prints the directory contents of a drive onto the screen (or printer - see below). The printout halts after 14 lines, continuing after any key depression.

The listing gives the file name, file type and length in domains. 1 domain = 256 bytes. A * by the file name indicates a write protected file. At the end of the listing the screen shows the amount of free space on the disk. An optional drive specification may be given.

Syntax

DIR	lists current drive contents
DIR A	lists drive A
DIR C	lists drive C

Examples

```
DIR
NAME          TYPE          LENGTH
INTRO        *BAS             18
NUMBER        BAS             13
PDATA         DAT             50
PINPUT        BAS              5
OK
```

Output to Printer
To output the DIR listing to printer, type the following:-

```
POKE&H6F,&HFE: DIR
```

The above command must be on one line as location &H6F is automatically reset after each BASIC line. DIR syntax on a multi-drive system is the same as the normal DIR.

Comments
Gives quick clear list of disk contents. Use before writing to an 'unknown' diskette.

Associated Keywords
INIT CONFIG KILL ASSIGN VERIFY SELECT

INIT INIT

Function
Prepares a disk for use or re-use. A fresh disk MUST be initialised before use. If a used disk is to be utilised the computer will ask: "DATA FOUND - USE?" Type Y to continue, 'N' or any key other than 'Y' stops the disk being wiped.

Syntax

INIT	initialise disk on default drive
INIT B	initialise disk on drive B

Examples

```
INIT
DISK CONTAINS DATA - USE? Y
OK
```

Comments
A write-protect label will stop a disk being initialised. Software write-protection is ignored by INIT!

The INIT process takes about 40 seconds on a standard 5.25" drive. Error messages are generated if there are problems with the media. An essential function, but use with great care on old disks. Be especially careful where 'DISK CONTAINS DATA' message appears when not expected!

Associated Keywords
CONFIG BACKUP

CONFIG CONFIG

Function
Sets up the system to use different types of disk drive. This information is stored on the disk by the INIT command. When a disk is booted or selected the computer resets to the configuration stored on the disk. The command is normally used before initialising a fresh disk. If the system only has one drive or all drives of the same type this command will not be used very often as in the normal course of events, any system disk will configure the computer on start up and no further intervention will be required.

CONFIG needs six parameters:-

1. Drive letter	A - D
2. Number of tracks	40 on a standard 5.25",
3. Sectors / track	10 for single density 5¼", 18 for double density
4. Number of sides	1 single sided, 2 = double sided
5. Step rate	1 = 12 ms 2 = 20 ms 3 = 30 ms†

6. Data Rate	D = 250 Kb/s (5.25" Double Density)
	S = 125 Kb/s (5.25" Single Density)

† *Seek errors or complete failure to read will occur if you attempt to use a faster Step Rate than that specified for your disk drive; always use parameter 3 for Step Rate where you are unsure.*

Syntax
CONFIG A,T,S,N,R,D

Examples

CONFIG A,40,10,1 ,3,S	sets up 40 tracks on Single Density 5,25" drive
OK	
INIT	
OK	
CONFIG A,40, 18,1 ,3,D	sets up 40 Tracks Double Density on 5.25" drive
OK	
INIT	

Comments
Mainly of use in systems where drives are changed over, not needed in most cases.

Associated Keywords
INIT

KILL KILL

Function
Deletes files from the disk, unless protected.

Syntax
KILL"NAME"
KILL"B:NAME"

Examples
As syntax

Comments
Use with care. Error messages are generated if the file is not found or software protected. Drive specifications can be used as a prefix in the file name. KILLed files cannot be retrieved!

Associated Keywords
DIR INIT CONFIG ASSIGN VERIFY SELECT

ASSIGN ASSIGN

Function

Renames and/or changes file protection.

ASSIGN has two functions which can be used separately or together. The rename is used to simply change the name of a file. The protection facility is used to write protect / unprotect a named file. A protected or sanctified file can be read by the computer but cannot be over-written. KILL will not erase a protected file. This is a valuable facility when important programs or data are to be protected.

Syntax

ASSIGN"OLD", "NEW"	changes OLD to NEW
ASSIGN"C:OLD", "NEW"	changes OLD to NEW on drive C
ASSIGN"NAME";S	Sanctify or 'protect' a file
ASSIGN"NAME";D	Desecrate or 'unprotect' a file
ASSIGN"OLD" ,"NEW";S	change name and protect

Note commas and semi-colons

Examples

As syntax

Comments

Always protect important program and data files. The command KILL cannot be used to remove a protected file.

N.B. ASSIGN does NOT protect against the INIT command!

Associated Keywords

KILL INIT CONFIG VERIFY SELECT

VERIFY VERIFY

Function

Enables or disables the disk verify operation. Normally the operating system re-reads the disk whenever any data is written to check that it has been recorded correctly. This gives greater security to the user as an error message is given if there are any difficulties.

The checking process takes a certain amount of time and slows down the disk access. Disk systems are very reliable and greater operational speed can be achieved if the checker is disabled. The

VERIFY D command turns off the re-read check; it can be restored by VERIFY E. VERIFY defaults to E.

Syntax

VERIFY D	disable disk verification, speed up access
VERIFY E	enable disk verification, slow access

Examples
As syntax

Comments
Do not use when handling important data; use in less vital applications where speed is more important.

Associated Keywords
INIT CONFIG DIR KILL ASSIGN SELECT

SELECT SELECT

Function
Engages the specified drive. Drives are numbered A to D. If an auto boot disk is in the drive it will boot normally. The specified drive is homed and reset. This command should be used after a change of disks.

Syntax

SELECT A	selects drive A
SELECT B	selects drive B

Examples
As syntax

Comments
SELECT should be used to access the required drive, especially after you have DIRed a drive other than the one containing your master program. If DRIVE NOT READY constantly appears, SELECT A to regain control! It is advisable to use this command after a change of disks.

Associated Keywords
INIT CONFIG DIR KILL ASSIGN VERIFY

COPY COPY

Function

Copies a named file from one disk to another. Although this is quicker in a multi-drive system it is possible to use a single drive. The system will prompt the user when to change disks etc. The original disk is called the "SOURCE DISK" and the new disk is called the "TARGET DISK".

Syntax

```
COPY"A:SOURCE","B:TARGET"          on a multi-drive system
COPY"A:SOURCE","A:T ARGET"         on a single drive system
```

Examples

```
COPY"A:FRED","A:BILL"                    for a single drive
INSERT SOURCE DISK <KEY>
INSERT TARGET DISK <KEY>
OK
COPY"A:FRED", "B:FRED"                   for multi drives
INSERT TARGET DISK <KEY>
OK
```

Comments

An easy way to make copies of valuable files. Can be used with a single drive system, but put a write-protect label over the source disk! Use BACKUP to copy an entire disk.

Associated Keywords

BACKUP

BACKUP BACKUP

Function

Copies an entire disk onto another disk. Can be used with single or multiple disk systems. The new disk is automatically initialised before use. This command is much faster in a multi disk system as no disk swapping is necessary. In a single drive system prompts are given to tell the user when to change disks.

Syntax

BACKUP

Examples

```
BACKUP <ENTER>
SOURCE DRIVE? A
TARGET DRIVE? B
DATA FOUND - USE? Y
OK
```

or

```
BACKUP
SOURCE DRIVE? A
TARGET DRIVE A
INSERT TARGET DISK <KEY>
DATA FOUND - USE? Y
INSERT SOURCE DISK <KEY>
INSERT TARGET DISK <KEY>
```

and so on until entire disk has been copied

```
OK
```

Comments

ALWAYS ensure there is a write-protect label on the source disk!

Associated Keywords

COPY

CREATE CREATE

Function

Used to create and prepare a serial or random access file. The length of file is specified on creation. The length is specified in domains, 1 domain = 256 bytes. The only limit on the size of file is the space available on the disk.

Syntax

CREATE"NAME" ,N

Examples

CREATE"DATA",48	creates a file 12k long
CREATE"A:HOLD",20	creates a file on drive A 5k long

Comments
Error messages are generated if there is insufficient space or if the name is already in use. The rules on file names are the same as the SAVE command. Be generous when creating data files or you may run out of file space. CREATE is not to be used to prepare a space for a BASIC program on disk - this is done automatically during SAVE.

Associated Keywords
FLUSH FILES OPEN CLOSE

FLUSH FLUSH

Function
Clears a data file for re-use under the same filename. This command completely erases a file and data cannot be recovered from a flushed file. The file may be addressed by name or by channel number.

Syntax

FLUSH"DATA"	flush file called DATA
FLUSH"B:NUMBERS"	flush file called NUMBERS on drive B
FLUSH#1	flush file connected to channel #1

Examples
```
FLUSH"DATA"
OK
10 FILES 2
20 OPEN#2,"DATA"
30 FLUSH#2
```

Comments
Files must be flushed if they are to be re-used - failure to flush a file can lead to odd errors in file handling programs. Use with care. Always use FLUSH"NAME" if possible - using the channel number method could lead to the loss of a valuable disk.

Associated Keywords
CREATE FILES OPEN CLOSE

OPEN OPEN

Function

This command is used to open a data file on a given input channel. The drive may be specified in the file name as a prefix. Only one file can be connected to a given channel at anyone time. Error messages are given if the channel is already in use or if the file is of the wrong type.

Syntax

OPEN#N,"NAME"	n = channel (1 to 8)
OPEN #N, "B :NAME"	

Examples

OPEN#5, "TEST"	connects a data file called TEST to channel 5

Comments

See chapter on File Handling.

Associated Keywords

CLOSE FILES CREATE FLUSH

CLOSE CLOSE

Function

A file handling command. When files are in use special memory buffers act as temporary stores. The CLOSE command ensures that all buffers are correctly written onto the disk.

A BASIC file handling program must always end with a CLOSE command. Failure to do this will result in loss of data on the file.

Syntax

CLOSE	CLOSEs all OPEN files
CLOSE#	CLOSEs only files on channel number 6

Examples

1000 CLOSE: END	normal syntax for CLOSEing files
900 CLOSE#6	closes file six for re-use
910 OPEN#6,"FRED"	

Comments
An error message is given if an attempt is made to CLOSE an un-open file. An important command in file handling; forgetting the CLOSE command at the end of a program is a very common mistake. If you accidentally hit BREAK during keyboard input, type CONT <ENTER> to put you back into the program. Should this fail, type CLOSE <ENTER>. This should save your data. **Unclosed files = lost data!**

Associated Keywords

CREATE	FLUSH	FILES	OPEN	CONT

FILES FILES

Function
Sets aside buffer space in the memory for data files. Each channel needs a 256 byte buffer in memory.

The system boots up with 1 file channel buffer as standard, more are specified using the FILES command.

A maximum of eight files may be specified. The FILES command should be used at the very beginning of a file-handling program. DO NOT USE after files have been OPENed.

Syntax

FILES N	N = number of channels required (1-8)

Examples

FILES 6	sets up 6 file channels
FILES 1	resets to 1 file channel

Comments
FILES should be the first command in a program. FILES 1 may be used to recover buffer space for program use if the file handling is no longer required.

Do not specify more files than are needed as this wastes memory. NEVER expand your FILES in the middle of a program as this will cause all the variables to CLEAR and CLOSE all files. File buffers are NOT written to disk after FILES so any information input will be lost! Program execution will continue (but will have become very confused!!). Never put the command FILES in a GOSUB routine - remember FILES issues a CLEAR statement so when your program hits the RETURN it won't know where to go and will give RETURN without GOSUB error!

Make sure you understand when and how to use this command if you are putting FILES anywhere other than at the start of a BASIC program

Associated Keywords

OPEN   CLOSE   CREATE   FLUSH   CLEAR

END#   END#

Function

Moves the file read/write pointer to the end of a serial file or random access record. This command is used to extend an existing file or record. In the case of a serial file the pointer is moved to the end of the current file.

Any new data is now written on the end of the existing file. This obviates the need to read the whole file before adding new data.

In Random Access mode the command allows individual records to be extended.

Syntax

END#N	(serial) N = channel number
END#N,%R	(random) R = record number

Examples

```
100 OPEN#1,"SDATA"                open file
110 END#1                         move pointer to end
120 INPUT"NEW DATA";ND$           add new data
130 PRINT#1,ND$                   to file
```

This replaces the usual method:

```
100 OPEN# 1, "SDATA"
110 IF END(1) THEN 140
120 INPUT#1,A$
130 GOTO 120
140 INPUT"NEW DATA";ND$
150 PRINT#1,ND$
```

These two routines are functionally identical. However the first is shorter and more convenient.

In random access:

```
100 OPEN#1,"RDATA"
110 END#1,%50
120 INPUT"NEW DATA";ND$
130 PRINT#1,ND$
```

Adds new data to record 50 (% record number not required) Instead of

```
100 OPEN# 1, "RDATA"
110 IF END(1) THEN 150
120 INPUT#1,%50,A$
130 INPUT#1,B$
140 GOTO 130
150 INPUT"NEW DATA";ND$
160 PRINT#1,ND$
```

Comments
Allows easy extension of existing data files.

Associated Keywords
RESTORE# DIM # FIND# IF EOF(N) INPUT# PRINT# FIND#

RESTORE# RESTORE#

Function
RESTORE# moves the read/write pointer to the start of a file or record. This allows the data to be re-read in the same way that the normal BASIC RESTORE allows data statements to be re-used.

This is essential if repeated searches of a file are required. In Random Access the pointer can be moved to the start of a particular record.

Syntax

RESTORE#N	N = Channel number
RESTORE#N,%R	R = Record number

Examples

```
100 OPEN#1, "PDATA"
110 IF END(1) THEN 170
120 RESTORE#1: REM RESET FILE
130 INPUT"NAME";M$
140 IF N$ <> M$ THEN 130
150 PRINT N$,T$
160 GOTO 110: REM GET NEXT NAME
170 PRINT"NAME NOT ON FILE"
180 CLOSE:END
```

Comments

Resets file data as the normal RESTORE resets DATA statements.

Associated Keywords

END#   DIM#   IF EOF(#)   INPUT#   PRINT#   FIND#

DIM# DIM#

Function

This command is an extension of the normal BASIC DIM statement. In addition to the normal functions it now allows the length of a random access file to be set. Only an open file may be dimensioned. The record length can be from 1 to 255. If longer records are required they can be used in twos or threes etc. If this command is not used records default to 128 bytes.

Syntax

DIM#(N,L)	N = channel number
	L = record length (1 - 255)

Examples

```
10 FILES 5
20 OPEN#5, "RANDOM"
30 DIM A (15),#(5,50),B$(10, 15),A$(20)
```

Comments

By using a suitable DIM#, records can be set up to use the minimum necessary space on the disk. DIM# will mix quite happily with other normal DIM statements on the same BASIC line.

Associated Keywords

END# RESTORE# IF EOF(#) INPUT# PRINT# FIND#

IF EOF(N) THEN

Function

This is an error trapping command. In the normal course of events an attempt to read past the end of a file will result in an error message. When the EOF(N) has been set the error message is bypassed and the program moves to the line number specified. If the line number is 0 the error message is reconnected. Each file channel may have its own trap.

Syntax

IF EOF(N) THEN 100	N = channel number (1 to 8)
IF EOF(N) THEN 0	0 restores error message

Examples

```
10 FILES 3
100 OPEN#3, "FRED"
110 IF EOF(3) THEN 150
112 IF EOF(2) THEN 500
114 IF EOF(1) THEN 300
120 INPUT#3,A$
130 PRINT A$
140 GOTO 120
150 PRINT"END OF FILE"
160 IF EOF(3) THEN 0 :REM TURN OFF TRAP
170 CLOSE
180 END
300 REM CONTINUE FOR TRAP ON CHANNEL 1
800 REM CONTINUE FOR TRAP ON CHANNEL 2
```

Comments

IF EOF(n) THEN should be near the start of the program. Up to eight such commands are allowed, one for each channel. As this command sets a flag for the error message routine, effectively telling it which line to go to when it finishes the file, it does NOT have to be constantly tested by your program, hence the advice to place it near the start. The best way to think of this command is as

'ON EOF(n) THEN GOTO xx' - we couldn't call it this for technical reasons!

(LINE)INPUT#

Function

This is used to pass data from a data file to a program in a similar manner to the normal INPUT routine. The # specifies which channel to use and the optional % specifies which record. Each successive INPUT# takes in the next piece of data on the file (c.f. READ & DATA). LINE INPUT# allows {, and ; to be used as data in a similar manner to the normal LINE INPUT command. Error messages are generated if the channel is not open.

Syntax

INPUT#N,A$	or	LINEINPUT#N,A$	(serial)
INPUT#N,B			(serial)
INPUT# N,A$,B,C			(serial)
INPUT#N,%R,A$	or	LINEINPUT#N,%R,A$	(random)
INPUT#N,%R,B			(random)

Examples

```
100 OPEN#1, "PDATA": REM SERIAL ACCESS EXAMPLE
110 IF EOF (1) THEN 150: REM GOTO LINE 150 WHEN END
          REACHED
120 INPUT#1,N$,P$: REM READS DATA FROM CHANNEL
          NUMBER 1
130 PRINT N$,P$
140 GOTO 120
150 CLOSE
160 END

100 OPEN#1 ,"RANDOM" : REM RANDOM ACCESS EXAMPLE
110 INPUT"RECORD NUMBER";R
120 INPUT#1,%R,N$,P$: REM NOTE '%' FOR RANDOM ACCESS
130 PRINT N$,P$
140 GOTO 110
```

Comments

See section on File Handling

Associated Keywords
PRINT# DIM# END# RESTORE#

PRINT# PRINT#

Function
PRINT#puts data onto a data file in the same way that PRINT puts data onto the screen. PRINT# puts each of the items of data after the last item. The # specifies the channel number and the option to give the record number. Error messages are given if the disk or file are write-protected. A separate PRINT# should be used for each item of data; USING may optionally be used to format data onto the file.

Syntax
PRINT#N,A$ (serial access)
PRINT#N, USING" ###. # #' ';C
PRINT#N,%R,D (random access)
N = channel number
R = record number

Examples
```
10 FILES 3
100 OPEN#3,"DATA"
110 ENDS#3
120 INPUT"NAME";N$
130 PRINT#3,N$
140 INPUT"PHONE NUMBER";P$
150 PRINT #3,P$
160 CLOSE#3
170 END
```

Comments
See File Handling.

Associated Keywords
END# RESTORE# DIM# IF EOF(#) INPUT# FIND#

Function

FIND is used to search serial and random access records for a specified string. In the case of a serial file when the string is found the next INPUT# will input the target string from its start.

In the case of a random access file the next INPUT# will input the start of the RECORD containing the string. A record can have several entries and the FIND command will locate whole records. This is very useful when searching random access records (See the NUMBER demonstration program). When a search is successful location $03FD is set to $FF (255); if unsuccessful it is set to $0.

The search starts from the current position of the file pointer. RESTORE#is used for a complete search of a file. In the FIND command the asterisk is treated as a wild character e.g: L*ST could be found as LAST,LEST,LIST,LOST or LUST. You can use as many asterisks as you like - PR****R MICRO**STEMS is quite legal and would be found if present on the searched file. However, * cannot be the first character in a search string, neither will it find an asterisk within the data file.

Random Access searches begin at the specified record number.

Syntax

FIND#N,A$	Serial Access syntax
FIND#N,%R,A$	Random Access syntax
	N = channel number
	R = record number
	A$ = search string.

Examples

```
10 OPEN#1,"DATA"
20 INPUT"STRING TO BE FOUND";C$
30 RESTORE#1
40 FIND#1,C$
50 IF PEEK(&H03FD)=0 THEN PRINT"NOT FOUND":END
60 INPUT#1,A$
70 PRINT A$
80 CLOSE: END
```

or

```
10 OPEN#1, "RANDOM"
20 INPUT"TARGET STRING";D$
30 FIND#1 ,%1 ,D$ : REM START SEARCH AT RECORD 1
```

```
40 IF PEEK (&H03FD)=0 THEN PRINT"NOT FOUND":END
60 INPUT #1 ,A$ : REM % NOT NEEDED AFTER A SUCCESSFUL
          FIND
70 PRINT A$
80 CLOSE:END
```

Comments
A powerful command, particularly with the inclusion of the wild character '*'. See File Handling.

Associated Keywords
END#   RESTORE#   DIM#   IF EOF(N)   INPUT#   PRINT#

BUILD   BUILD

Function
This command is used to create an executive file onto disk. The named file must not already exist. A file is created to hold the text to be executed. The file may be up to 255 characters in length, but files can be chained together using DO.

The prompt 'TYPE 255 CHARS' appears, after which text is entered as required including all carriage returns etc. The <BREAK> key is used to exit this mode and write the file to disk. N.B

All characters are accepted by this routine including CR, CLEAR, BACKARROW etc and are counted in the 255 allowed. When the file is later executed they will have their normal effect.

Syntax
BUILD"NAME"
BUILD"C:NAME"   for a multi drive system.

Example
```
BUILD"EXAMPLE" <ENTER>
TYPE 255 CHARS
LOAD"PROG" <ENTER>
LIST
RUN
<BREAK>
```

The above example would LOAD a program called PROG, LIST it then RUN it, all without operator intervention.

To activate the program, type:-
DO"EXAMPLE" <ENTER>.

Comments
Saves itself onto disk using your chosen filename and the extension BLD. Use the DO command to activate a BUILD file. Can also be tied in with the BOOT command to give long sequences on initial boot of disk.

Associated Keywords
DO BOOT

DO DO

Function
Executes a file created by BUILD. Enables the contents of the file to be treated as keyboard input. This command may be used within BUILD files to execute other BUILD files and thus extend the command string.

Syntax
DO"NAME"
DO"C:NAME" for a multidrive system

Comments
Used to activate a file composed by the BUILD command.

Associated Keywords
BOOT BUILD

BOOT BOOT

Function
This command allows a disk to automatically carry out a predetermined instruction when the disk is either booted or selected. Only one command can be given. However as BOOT can RUN a selected program or DO an executive file this is no real restriction. The command BOOT alone cancels the boot format.

Syntax

BOOT : RUN"INTRO"	RUNs the program named INTRO on boot
BOOT	cancels prior boot instruction

Examples

```
BOOT:RUN"MENU"
BOOT:LOADM"CODE"
BOOT:RUNM"TOOL"
```

Comments

This command can be used with applications software as the user does not need to LOAD or RUN a program. An autoboot disk can run a menu program which can control a suite of routines. Combined with BUILD/DO, it becomes a very powerful command. Not intended for a direct PRINT message as BOOT removes spaces!

Associated Keywords

DO BUILD

Record Specifiers >< ** <>

In Random Access a record is normally addressed by number. INPUT#2,%N,A$: where N is a numeric variable.
However it is often more convenient to move from record to record without using a record number. In DELTA this is possible using the special record specifiers > < and *, which allow the program to step forwards OR backwards through a file.

* accesses the last record used, i.e if the last record used was 50 then * will move to record 50.

< accesses the file before the last record used, i.e if the last record used was 50 then < will move to record 49.

> accesses the file after the last record used, Le if the last record used was 50 then> will move to record 51. This is useful for stepping through records.

These specifiers can he used in any command that uses a record number. Using '<;', you can even step backwards automatically through the file without complicated BASIC statements. However, note that FIND will always search forwards from the record set.

Example

```
INPUT #5,%*,A$
FIND#7,%>,C$
```

Machine Code Files

An interesting feature of DELTA is its ability to store a specified area of memory onto disk. This enables the user to store machine code programs, graphics screens or binary data directly. The SAVEM command is used to transfer the relevant area of memory to disk. The instruction takes the form:

SA VEM"NAME" ,START ADDRESS,END ADDRESS.

where NAME is the chosen file name, START ADDRESS is the first location of the required memory area and END ADDRESS is the final location. The system will automatically create a file of suitable size. The target addresses may be in decimal, hexadecimal (using the &H prefix) or octal (using the &O prefix).

Example

```
SAVEM"CODE", 17650,18776
SAVEM"SCREEN" ,&H0400,&H05FF
SAVEM"EIGHT" ,&0777,&02700
```

For most applications the hexadecimal value will be the most useful. In a program the start and end may be specified by computed (decimal) variables e.g

SAVEM"CODE",A,B*6 etc.

Example: saving the contents of the text screen to disk. The screen occupies hex locations $0400 to $05FF. To save the screen type:-

```
SAVEM"SCREEN",&H0400,&H05FF
```

The directory should now show the new file; this file will be shown as a CMD (command) type. Now enter:

```
CLS
LOADM"SCREEN"
```

This will clear the screen and recover the original data. The same technique is used to save programs written in machine code.

As shown above, machine code files are recovered from the disk using the LOADM command; this takes the form:
```
LOADM"NAME" or LOADM"NAME",ADDRESS.
```
The first command loads the file to the area in the memory from which it came. Optionally the data may be wanted at some other location. In this case the address specified is the new start address.

Example
```
SAVEM"TESTS",&H0400,&H04FF : REM TOP HALF OF TEXT
SCREEN
CLS
LOADM"TESTS" : REM RETURN TO ORIGINAL LOCATION
CLS
LOADM"TESTS",&H0500 : REM RETURN TO BOTTOM HALF OF
TEXT SCREEN
```

An additional command allows a machine code program to be loaded and executed. This command assumes that the programs execution address is the same as its start address. Programs should be written with this in mind;
The command is:-
```
RUNM"NAME"
```

If the execution address is not at the start of the code the following command can be used:-
```
LOADM"CODE":EXEC START
```
where CODE = filename, START = execution address.

Associated Keywords

SAVEM	RUNM	LOADM	EXEC

Drive Selection

In a multi-drive system it is possible to shift between drives without using the SELECT command. This is done by using a 'Drive Prefix' and a filename. The prefix consists of a drive letter (A,B,C or D) and a colon.

This method of selection is usually much more convenient.

Examples

LOAD"B:NAME"	will select drive B before loading the program.

RUN"D:PHONE"	looks on drive D for a program called PHONE and run it.
DIR B	prints directory of drive B.

Note:- That if you SELECT a drive which does not exist, whether via SELECT or DIR or RUN"C:xxxx", DELTA will remain logged onto that drive even though the 'Drive Not Ready' or similar error message will have been generated. To get back to your master drive, simply type DIR A.

Program Files

A program file is used to store a computer program when not in use. The size of file required will vary from program to program, however in DELTA the DOS can create a file of the correct size using the SAVE command. Programs are recovered from the disk using either the RUN or LOAD commands.

Loading and Running

In order to get a program from the disk into the computer workspace, we use the LOAD command.

Example

```
LOAD"PROGI"
```

will look for the program named PROGI on the disk and load it into the workspace. (NB. this will erase any program already resident in the workspace).

PROGI may now be RUN, LISTed, or altered. Usually we want to LOAD and RUN a program; this is done with the BASIC RUN command.

Example

```
RUN"PROGI"
```

will LOAD and RUN the program called PROGI all in one action. This command can also be used within a BASIC program, allowing one program to call another. Note the use in the next example of CHAIN - used to retain program variables:-

```
1000 INPUT"Type A for Accounts, S for Stock, E for END";Y$
1010 IF Y$ ="A" THEN RUN"ACCT"
1020 IF Y$ ="S" THEN CHAIN"STOCK"
```

```
1030 IF Y$<>"E" THEN 1000
1040 END
```

This technique allows a simple 'MENU' program to access a suite of related programs. Each of the related programs would RUN and MENU again when they are finished. The advantage of this approach is that a suite of simple programs can be used for a particular application. It is always easier to write several simple programs than one long complex program. DATA is passed between programs using DATA files.

Example

```
SAVE"PROG3"
```

will find a space on the disk large enough for the program, create a file called PROG3 and then store the program. If there is not enough space on the disk to store the program an error message will be generated.

If an existing program has been corrected or amended, the SAVE command should be used to store the new version onto the existing file. NB. This erases the old version.

Storing Programs

The SAVE command is used to store a program which is in the workspace onto the disk. When a program has just been written, or a new copy is required on a fresh disk, use SAVE.

Example

```
SAVE"PROG3"
```

If you are working with a valuable program, and want to be 100% sure of not losing both versions due to a silly error, use this slightly longer sequence using the ASSIGN command.

```
SAVE"HOLDER"
LOAD"HOLDER" TO CHECK NEW VERSION HAS BEEN SAFELY
STORED.
LIST
KILL"PROG3"
ASSIGN"HOLDER","PROG3" GIVES THE NEW PROGRAM THE
ORIGINAL NAME.
```

Data Files

Data files are the method by which a computer can store information over a long period of time. This information can be recalled by either the program that produced the data, or by any other suitable program. In some applications a single program will put data onto a file and then recover it at a later date. However, it is more common for a suite of programs to use a common set of files. One program would be used to collect and file the data, while other programs would process and present the data as required.

In order to make use of files, we need to know how to write data onto a file, how to locate filed data, and how to read the data from the file. In many ways the principles are the same as an office filing system using filing cabinets and ordered files. Each disk is like a small filing cabinet which can hold a number of files. A particular disk may only hold one very large file or many small ones. Each file must be given a unique name when it is created; the computer does not allow two files on the same disk to have the same name. These filenames allow the computer to locate the correct block of information on the disk.

Before any file can be used for data it must be created. The creation of a file formats the file correctly for data use and ensures that no spurious data gets on the files. When a file is created its length must be specified. This requires a generous over-estimate of the maximum amount of information to be put on the file. One domain of the file will hold 256 characters. (The length of Random Access files will be discusses later).

Let us create a file called TEST/F, fifty domains long. Type in the following:-

```
CREATE"TEST/F",50
```

We now have a file ready for a program to use.

Programs communicate with data files via the input/output channels. This system allows the program to use up to and including eight channels simultaneously. The channels are numbered from onr to eight. In use each file is allocated a separate channel by the program. All the channels operate in the same way and none has any greater significance than the others.

A file is connected to a particular channel using the OPEN command.

Example

```
30 OPEN#3,"TEST/F"
```

This command will then allow the computer to access the file TEST/F via channel 3.

Data is sent to the file using the PRINT# command and is recovered from the file using INPUT# When the file is opened, the computer always look at the very start of the file, each PRINT# and INPUT# command works the computer along the file from the beginning unless otherwise instructed (eg using the END# and RESTORE# commands).

When a file is no longer required by a program it MUST be closed. This is because each channel uses a temporary store called a disk buffer. The CLOSE instruction makes sure that all this information is safely stored onto the disk. Channels may be CLOSEd individually – CLOSE#3 will close channel 3 only.

Frequently it is necessary to close all open channels at the end of a program and this can easily be achieved by putting:-

```
1000 CLOSE
```

We will now show how to store and recover some simple numbers and strings. For these programs use a freshly formatted disk

PLEASE DO NOT EXPERIMENT WITH YOUR MASTER DISK!

1. First CREATE a suitable data file called STEP/F, 30 domains long

   ```
   CREATE"STEP/F",30
   ```

2. We will now put some data onto this file using a simple program.

```
10 REM FILE PRINT PROGRAM NUMBER ONE
20 CLS:OPEN#1,"STEP/F":REM ASSIGN FILE TO CHANNEL 1
30 A$="TEST STRING":REM CREATE A STRING
40 FOR J=1 TO 30
50 B$=A$+STR$(J):REM ADD A FIGURE TO EACH A$
60 PRINT#1,B$:REM PUT THE STRING ONTO THE FILE†
70 PRINT#1,(J*5):REM PUT A NUMBER ONTO THE FILE†
80 PRINT B$,J*5:REM DISPLAY DATA ON SCREEN
90 NEXT J
100 CLOSE
110 END
RUN<ENTER>
```

† *when using data files, use a separate PRINT# command for each item of data.*

We have now put some simple test data onto a file. Always remember that data is stored on the file in exactly the same order as it was created by the program, so the recovery program must take off the data in the same order. Our first program laid down thirty strings alternated with thirty numbers (i.e. string, number, string, number ....). An equally simple program is used to recover the data. Our new program must look in the same order as the creating one.

RUN the above program, NEW the workspace and enter the program below.

```
10 REM FILE INPUT PROGRAM#1
20 OPEN#1,"STEP/F" : REM PUT FILE ON CHANNEL 1
30 FOR J = 1 TO 30
40 INPUT#1,C$,K : REM TAKE DATA FROM FILE
50 PRINTC$,K : REM DISPLAY ON SCREEN
60 NEXT J
70 CLOSE
80 END
RUN <RETURN>
```

We have now managed to save and recover data from a disk. Try a few variations on the programs above and see what happens. Try changing the A$ to be something longer or more useful!

If you wish to wipe out the data on the file between experiments, include the following line in the first program

```
25 FLUSH#1
```

Now attempt to read more data from the file than was written by changing line 30 to '1 to 50' and see what happens. Try putting a very large number of strings on the file, etc, always noting carefully what happens.

Before proceeding to the next chapter, try to write your own programs based on these two simple examples. Remember the following points

1. Be generous when creating files
2. Data files must be flushed if needed for fresh data
3. Each file must be opened on a separate channel (1-8)
4. Each different item of data needs a separate PRINT# statement- it does not matter with INPUT#
5. FILES MUST BE CLOSED. **Unclosed file = LOST DATA**
6. Take data from the file in the same order that it was put down.
7. On start-up, only ONE file channel is available. Use the FILES command if more are needed (up to 8 allowed). The FILES

command should be at the very start of a program, straight after your REM statement.

The following chapters will show how to manipulate and modify files. Be sure you have understood so far. If not - keep experimenting. if you are really stuck, phone in to us Monday evenings 7 - 9pm. during our Customer Service Session.

More on Serial Files

The type of file introduced in the last section is known as a serial file. In this section we shall see how these files can be manipulated using the special BASIC file commands which have been incorporated to make file searching and amendment much easier.

When a file has been opened, the 'file pointer' always points at the beginning of the file. If we want to add something onto the end of the file, this pointer must be moved to the free space at the end of the file. This could be done by reading the whole file until we reach the end via the IF EOF(l) THEN statement as follows:-

```
100 OPEN#1, "STEP/F""
110 IF EOF (1) THEN 140
120 INPUT #1 ,A$
130 GOT0120
140 REM: REST OF PROGRAM
```

This method is quite satisfactory, but rather slow and clumsy, especially if the file is a long one. A neater method is to use the END command.

```
100 OPEN# 1, "TEST IF"
110 END#1 : REM MOVES POINTER TO FREE SPACE AT THE
      END.
```

Note that the relevant channel number must appear after the END# statement.

The END# command is obviously neater and much faster in operation than a searching loop. Note that an error message is given if the channel is CLOSEd.

This technique is used to extend a serial file, allowing fresh data to be added to existing data. The following example shows how this could be used in a very simple phone number file (remember to CREATE the file first).

```
10 REM PHONE NUMBER INPUT PROGRAM
20 OPEN#1,"PHON/F"
```

```
30 END#1 : REM FIND END OF FILE
40 INPUT"SURNAME"; N$ : REM ENTER NAME
50 IF N$="END" THEN 110 : REM CHECK FOR END
60 INPUT"Phone Number";P$ : REM ENTER PHONE NUMBER
70 INPUT"O.K.";Y$ : IF LEFT$(Y$,1) <>"Y" THEN 40 : REM
   VERIFY INPUT
80 PRINT#1,N$ : REM PUT DATA ON FILE
90 PRINT#1,P$ : REM PUT DATA ON FILE
100 PRINT:PRINT:GOTO40
110 CLOSE#1 : REM CLOSE FILE
120 END
```

Every time this program is run it will add new names and numbers to the end of the file. Try this program and build up a small file of numbers and names - make some up if you have no friends! We will use this file in the next few examples.

Having created a file we want to enter a name and have the computer find the phone number. In this case we would always want to start our search at the start of the file, not at some point elsewhere. We can start the file at the beginning by using the RESTORE# command. This automatically moves the file pointer to the beginning of the file. The IF EOF (n) statement is used in the event of a piece of data not being located. The next example demonstrates the use of these commands.

```
10 REM PHONE NUMBER FINDER VERSION ONE
20 OPEN# 1,"PHON/F" : REM OPEN FI LE
30 RESTORE#1 : REM PUT POINTER TO START OF FILE
40 IF EOF(1) THEN 100
50 INPUT"NAME";M$ : IFM$= "END" THEN120
60 INPUT#1,N$,P$ : REM TAKE NUMBER FROM FILE
70 IF N$<>M$ THEN 60 : REM CHECK FROM MATCH
80 PRINT'THE NUMBER IS";P$
90 PRINT:PRINT:GOTO50
100 PRINT"NAME NOT ON FILE":PRINT
110 GOTO30
120 CLOSE: END
```

Note:- The IF EOF(l) THEN statement can go anywhere before the INPUT#l statement. If the IF EOF(l) THEN is omitted, an END OF FILE error will be generated when the INPUT# reaches the end of its data. The IF EOF(l) THEN can be turned 'off by using the command IF EOF(l) THEN 0. This reconnects the END OF FILE message.

A faster and more flexible search of a serial file can be done using the FIND routine. This searches a file for a particular string. If this string is found, the next INPUT#will take the wanted string. The FIND command sets a flag at memory location $03FD hex. A '255' signifies a success, 0 signifies failure. The search starts wherever the file pointer has been left. It is usual to RESTORE# the file before using a FIND. The next example uses FIND to locate a phone number.

```
10 REM PHONE NUMBER FINDER VERSION TWO
20 OPEN#1,"PHON/F" : REM OPEN FILE
30 PRINT:INPUT"NAME";M$ : IFM$="END" THEN 110
40 RESTORE#1: REM START FROM BEGINNING
50 FIND#1,M$ : REM SEARCH FOR M$ ON CHANNEL 1
60 IF PEEK (&H03FD)=0 THEN 100 : REM CHECK FOR
   SUCCESSFUL SEARCH
70 INPUT#1,N$,P$
80 PRINT"THE NUMBER FOR"N$;" IS ";P$
90 GOTO30
100 PRINT"NAME NOT FOUND": GOTO30
110 CLOSE:END
```

This version will operate much more quickly on a long file than the first example. Try extending your original file as much as possible then search for a number known to be at the end of the file. The FIND command sets up the INPUT# pointer to the start of the target string.

FIND also has another invaluable feature - the wild character *. This is used when the exact structure of the target string is not known. An asterisk is used for any unknown character in a search. The asterisk cannot be the first search character. For example:- A FIND of L*ST would locate LAST, LEST, LIST, LOST or LUST on the file J***SON would locate JACKSON or JOHNSON.

The use of an asterisk as a wild character enables half-forgotten data to be retrieved with ease END#, RESTORE# and FIND allow programs to be short and effective. Before proceeding, experiment with the examples given.

Serial files are easy to use and make the best use of file space, but they are difficult to amend because the data is packed up onto the disk. Removing or adding data in the middle of a file is a problem. We can overcome this problem by using a temporary holding file and rewriting the original. The next example will show how we would delete a name from the file.

```
10 REM FILE DELETION PROGRAM
20 FILES2
30 CREATE"HOLD",30: REM 'HOLD' MUST BE SAME LENGTH AS
   PHON/F
40 OPEN#1,"PHON/F"
50 OPEN#2,"HOLD"
60 INPUT"NAME TO BE REMOVED";M$
65 IF EOF (1) THEN 120
70 INPUT #1 ,N$,P$
80 IFN$ = M$ THEN 110 : REM DO NOT RECORD ONTO HOLDI
NG
   FILE
90 PRINT#2,N$ : REM STORE ON HOLDING FILE
100 PRINT#2,P$ : REM N.B. SEPARATE PRINT#
110 GOTO70
120 CLOSE
130 KILL"PHON/F"
140 ASSIGN"HOLD","PHON/F";D
150 END
```

This program shows the general principle. Expansion of line 80 into more lines would allow amended data etc, to he added to the holding file.

Another method of manipulation is to put all the file data into an array. Modify the array and then put it back onto the file. This is quicker and more flexible, but if the file is very large there may be problems with memory size!

```
5 FILES1
10 REM AMENDMENT BY ARRAY PROGRAM
20 DIMN$(150), P$(150) : REM SET ACCORDING TO SIZE OF
FILE
30 OPEN#1,"PHON/F"
40 IF EOF(1) THEN 90
50 N=0
60 INPUT#1 ,N$,P$
70 N=N+1: N$(N)=N$:P$(N)=P$
80 GOTO 60
90 IF EOF(1) THEN 0 : REM RESTORE END OF FILE MESSAGE
100 RESTORE#1 : REM RESTORE POINTER
110 REM MODIFY ARRAYS
120 I NPUT"NAME" ;M$: IFM$=" END"THEN 180
130 INPUT"NEW NUMBER";P1$
140 FORJ=1 TO N
150 IFN$(J)=M$ THEN P$(J)=P1$: REM ALTER NUMBER
```

```
160 NEXT J
170 GOT0120 : REM LOOP BACK FOR ANOTHER NAME
180 REM PUT NEW ARRAYS BACK ONTO FILE
190 FLUSH#1 ; REM WIPE ORIGINAL FILE CLEAN
200 FOR J = 1 TO N
210 PRINT#1 ,N$(J)
220 PRINT#1 ,P$(J)
230 NEXT J
240 CLOSE
250 END
```

This chapter and its examples have shown how to manipulate serial files. The examples show only a fraction of the potential of this kind of file. Remember that you can use up to eight files simultaneously As always, keep experimenting with variations on the supplied programs. Try and write your own programs and do not be disheartened when a silly error erases your whole file – it happens to everyone at times!

Remember to CLOSE files after use and be careful about the use of FLUSH which will wipe a file clean beyond recovery. Keep copies of valuable files, using either COPY or BACKUP.

Random Access Files

As we have seen, serial files arc strings of data in one long sequence. This is a compact method of data storage, but as already shown, inconvenient to change or update easily. A more flexible type of file is the Random Access file. In this sort of file each item or group of items is stored in separate records. Each record can be recovered or modified independantly of all the other records.

The industry standard length of each record is 128 characters, and the system automatically assumes this record length. However DELTA allows this to be altered with great simplicity (as wi1l be explained later).

Random Access files are created in the same way as a serial file. Using the standard record length, each disk domain stores 2 records, thus making the calculation of file length easy. Below is an example file created la hold l00 records.

CREATE" RAND/F",50   50 domains for 100 records

Random Access files are accessed using an extension of the PRINT# and INPUT# statements.

For example:-

```
PRINT #5, % 15," EXAMPLE"
```

puts the word 'EXAMPLE' onto channel number 5, record 15.

```
INPUT #5, % 15,A$
```

would recover the string from the same channel and record. Note the punctuation.

The '%' sign is used to denote the record number in the file. The next two examples show how to store and recover data from a Random Access file.

```
10 REM PRINT TO RANDOM ACCESS FILE
20 OPEN#1, "RAND/F" : REM OPEN FILE
30 FLUSH#1 : REM CLEAR FILE
40 A$="TEST STRING"
50 FORJ=1TO50 : REM PUT 50 ITEMS ON FILE
60 B$=A$ + STR$(J) :PRI NTB$
70 PRINT#1,%J,B$ : REM PRINT TO FILE 1 RECORD J
80 NEXT J
90 CLOSE : REM CLOSE FILE
99 END
RUN
```

```
10 REM INPUT FROM RANDOM ACCESS FILE
20 OPEN#1,"RAND/F"
30 FOR J=1TO50
40 INPUT#1,%J,K$
50 PRINTK$
60 NEXT J
70 CLOSE
80 END
RUN
```

Two things will be evident when these examples have been tried. Firstly, the BASIC syntax is no longer or more difficult than the syntax used within the serial files and secondly the program runs slightly slower than the serial examples: This slower operation only occurs when reading the entire file; access to individual records is much quicker. Make the following changes to the second example

```
30 INPUT"RECORD NUMBER";J:IFJ<1 OR J>50 THEN 30
60 PRINT:GOTO30
```

This will allow you to easily pick out a single record.

Records may contain more than one item of data, as each record can be used as a separate serial file In this mode, only the first item of data uses the special % notation. This can be demonstrated by changing our first example as follows:-

```
75 PRINT#1,(J*J) : REM NOTE ONLY THE FIRST PRINT# HAS A
    '%'
80 PRINT#1, "EXTRA DATA" : REM STILL USE ONE PRINT#
    PER ITEM
```

The data can be recovered using these new lines in the second example:-

```
55 INPUT#1,A,B$ : REM NOTE NO '%' SIGN
56 PRINTA,B$
```

This method of using the records allows related data to be kept together.

The standard record length of 128 characters is a fair comprom-ise for a variety of needs. Many systems do not provide an easy method of altering record length for this reason. However, in some cases 128 characters is very wasteful - why use 128 characters when the record will never contain more than 50. In other cases, 128 may be restricting and lead to awkward programs. DELTA allows the record length to be altered using the DIM statement after the file has been OPENed. The statement takes the form

```
DIM#(5,60)
```

where file '5' would be set to a record length at' 60 characters instead of 128.

Only an OPEN file may have its length set and different files may have different record lengths.

The record length may be between 1 and 255 characters; longer records are produced by using records in twos or threes etc.

For example:-

```
100 DIM#(1,250)
110 FOR J=1TO100 STEP2
120 PRINT#1,%J,A$
130 PRINT,B$
.
.
.
200 NEXT J
```

This type of routine would give an effective record length of 500 (2 x 250). When using variable length records, do not set the record any larger than actually necessary. A good tip is to store the actual record length on record zero, as follows:-

```
10 FILES5 : OPEN#5,"RAND/F"
20 INPUT"RECORD LENGTH";R
30 DIM#(5,R)
40 PRINT#S,%0,R
etc
```

When reading a Random Access file, the DIM MUST be the same value as when the file was written. Otherwise the data will not be found except on record zero. Using our record zero trick, we start our recovery program:-

```
10 FILES 3 : OPEN#3,"RAND/F"
20 INPUT#3,%0,R
30 DIM#(3,R)
etc
```

As a good general principle, use record zero in this way and start storing data from record 1.

Updating a Random Access file is very easy; simply write the record again. However, if each record has more than one item the whole record must be rewritten. It is good practice to erase the old record with nulls before reuse. The following routine shows how this can be done.

```
1000 NC$=STRING$(L-I,O) : REM L = record length
1010 PRINT#N,%R,NC$ : REM N = channel number, R = Record
          number
1020 RETURN
```

Random Access files are much easier to use when information is frequently revised or amended. However, remember that a Random Access file uses much more disk space and is slower if the entire file has to be read at one time.

The main difficulty with Random Access files is remembering on which record a particular item of data was stored. DELTA overcomes this difficulty using a highly advanced indexed file system.

Indexed Random Access Files

The indexed Random Access file overcomes the problem of locating data in a very large Random Access file. The system involves a small serial file which acts as an index for one or more Random Access files. A simple example would be customer files in a business. The index file would hold only the customer's name and record number. The record on the Random Access file would hold all the details on the customer. The secret of this type of file is to put as little as possible on the index.

This technique combines the advantages of both types of file; the index file may be searched using the FIND command and the record can be updated with ease.

The following examples show how an address file could be maintained using this approach. We will need two files.

```
CREATE"ADDR/F",60 : REM RANDOM FILE
CREATE"INDX/F",10 : REM INDEX FILE
5 FILES 2
10 REM INDEX EXAMPLE PROGRAM ONE
20 OPEN#I,"ADDR/F"
30 OPEN#2,"INDX/F"
40 N= 1: REM READ THROUGH INDEX AND FIND TOTAL No
          RECORDS
50 IF EOF(2) THEN 80 : REM BYPASS END OF FILE MESSAGE
60 INPUT#2,N$,M : REM INPUT NAME AND RECORD No
70 N=N+I :GOTO60: REM N=No OF RECORDS PLUS ONE
80 IF EOF(2) THEN 0 : REM TURN END OF FILE MESSAGE
85 RESTORE#2:END#2
90 DIM#(1,60) : REM SET RECORD LENGTH
100 PRINT#1,%0,60: REM STORE RECORD LENGTH
110 PRINT:INPUT"SURNAME AND INITIALS";N$
120 IF N$="END" THEN 260
130 INPUT"NUMBER AND ROAD";R$
140 INPUT"TOWN";T$
150 INPUT"COUNTY AND POSTCODE";C$
160 INPUT"PHONE NUMBER";P$
170 INPUT"ALL CORRECT';Y$:IFLEFT$(Y$,I )<>"Y"THEN
          110 : REM VERIFY DATA
180 REM PUT DATA ONTO FILES
190 A$=N$+R$+ T$+C$+P$:IFLEN(A$)<S5 THEN210 : REM
          CHECK LENGTH OF DATA
```

```
200 PRINT"TOO LONG":GOTO110
210 PRINT#2,N$:PRINT#2,N: REM NAME AND RECORD ONTO
          INDEX FILE
220 PRINT#I,%N,N$: REM NAME ONTO RECORD N
230 PRINT # 1, R$: PRI NT # 1, T$ :PRINT # 1 ,C$: REM ADDRESS
          ONTO RECORD N
240 PRINT#I,P$: REM PHONE NUMBER ONTO RECORD N
250 GOTO110
260 CLOSE
270 END
```

This program will also add new names onto the end of existing files. If you want to use this program for a large number of names and addresses, create larger files before you start. The second example shows one method of locating the data.

```
10 REM ADDRESS FINDER PROGRAM
20 FI LES 2 :OPEN # 1," ADDR/F" :REM OPEN FI LE
30 OPEN#2,"INDX/F"
40 INPUT#I, %0,R
50 DIM#(I,R): REM SET RECORD LENGTH
60 PRINT:INPUT" NAM E" ;M$: I FM$=" END"THEN 180
70 RESTORE#2: REM RESET INDEX FILE
80 FIND#2,M$: REM FIND NAME ON ON INDEX
90 IF PEEK(&H03FD)=0 THEN PRINT"NAME NOT
          FOUND" :GOTO60
100 INPUT#2,L$, R: REM GET RECORD NUMBER
110 REM RECOVER RECORD
120 INPUT # 1, % R,N$, R$,T$,C$,P$
130 PRINT: PRINTN$:PRINTR$: PRI NTT$ :PRINTC$
140 PRINT:PRINTP$
150 FIND#2,M$: REM CHECK IF NAME OCCURS AGAIN
160 IF PEEK(&H03FD) <>0 THEN 100
170 PRINT:PRINT:GOTO60
180 CLOSE
190 END
```

These programs can be easily modified to take in alternative or additional data. Even in this simple form, it demonstrates the power of the system of filing. In a more complex form, the index file might also contain the name of the Random Access file, allowing the index to call in fresh files. For example:-

```
300 INPUT#I,N$,F$,R: REM NAME, FILE AND RECORD
310 CLOSE#I: REM CLOSE EXISTING FILE
```

```
320 OPEN#2,F$: REM OPEN NEW FILE
```

The second program can be modified so that instead of printing the address, it inputs an amended address. To do this, delete lines 110-160 and insert an input routine as shown in the first program.

Another requirement of this technique is to create more than one index file. In our example, we indexed by name; however, we could simultaneously index {on separate files} by town or phone number. In a library program, one might index by author, publisher or title. When creating files, always give some thought to how the data is to be used. In a multi-disk system, keep the control program and index(s) on the main drive (A) and the large Random Access files on a separate drive. Always keep each index file down to a minimum of information. This will allow the searches to be as fast as possible. The index files are amended using the methods discussed and demonstrated in the chapter on serial files.

We have now examined the methods by which data files can be used within programs to store and retrieve data. As a general rule it is easier to use a suite of programs sharing files than to use a complicated general purpose program. Each small program is easier to write, takes up much less memory and loading is quicker. When writing data handling programs test them extensively with dummy data before putting valuable data onto disk.

ALWAYS MAKE COPIES OF VALUABLE DISKS.

Data can be lost very easily by silly programming errors (like not CLOSEing a file). Do practice writing file-handling programs as once the techniques are mastered, it is a fast and reliable way of storing a large amount of data. You may use up to eight files at a time. If this is not enough, some files should be temporarily CLOSEd and others OPENed when they are needed.

When writing data onto existing files, make sure you have moved to the free space at the end of the file (use END). In the case of a Random Access file, make sure that the record is empty. The next example shows a method of finding the first empty record on a Random Access file. This will work on any Random Access file regardless of its contents.

```
190 OPEN#l, "ADDR/F"
200 IF EOF (1) THEN 250
210 N= 1
220 INPUT#5,%0,R : DIMFunction(5,R) : REM SET RECORD
          LENGTH (OPTIONAL)
```

```
230 INPUT #5, %N,A$
240 N=N+1 :GOTO230
250 REM N NOW CONTAINS FIRST EMPTY RECORD No.
260 IF EOF (1) THEN 0
```

Another dodge is to record the highest used record number along with the record size onto record zero.

```
2010 PRINT#5,%0,R:PRINT#5,H : REM R=RECORD SIZE,
         H= HIGHEST RECORD IN USE
```

and to recover

```
90 INPUT#5,%0,R,H
100 DIM#(5,R): N=H+I: REM N= FIRST FREE RECORD
```

Transfer of Programs from Tape to Disk

Once your DELTA system is up and running, you will undoubtedly want to transfer your cassette programs to disk. If they are in BASIC, simply follow the steps below.

1. Ensure that the DELTA system is plugged in and running.
2. Type CLOAD "filename" where 'filename' is the name of the program, then hit ENTER.
3. Press PLAY on your cassette recorder and the program will load.
4. Once the program has loaded, save the program to disk using the syntax SAVE "filename" where 'filename' is your chosen name for the program.
5. Type RUN "filename" and the program will load from disk and autorun.

The above procedure will work as long as there are no machine code inserts in the program. If the program uses machine code routines (look for EXEC in the program), it may not be possible to run them from disk without a lot of program conversion. Here's how to find whether a BASIC/machine code program will work correctly:-

5. Follow steps 1 to 4 above.
6. Type LOAD "filename" to load the program from disk. DO NOT RUN IT YET.
7. Open the disk drive door and remove the diskette.
8. Now RUN the program. Pay particular attention to the disk

drive - if the red disk access light comes on during the program, the machine code part of the loaded program has overwritten the part needed by DELTA. Switch off your DRAGON and only use the program from tape!

9. If the program stops unexpectedly, DELTA has won the fight and you can only use the program with the DELTA card disconnected!
10. If the program runs all the way through with no unexpected occurrences AND you are able to use DELTA normally afterwards it is safe to assume that all is O.K. and that you can treat the program as a normal BASIC program.

Transferring Machine Code Programs

Transfer of machine code programs from tape to disk is not possible unless you have or can obtain the following information for the program concerned:-

(a) Area of memory where the program is located.

(b) Entry address of program.

The above information is required by the SAVEM command when saving the code to disk - without it saving is not possible. It is worth looking closely at the documentation supplied with the software to see if the information is given.

Assuming that you have the information required, save the code to disk using the syntax normally used by the SAVEM command - see relevant section. Retrieve the program with RUNM.

Transferring Cartridges to Disk

The games cartridges and the DELTA disk system use the same memory locations so it is not possible to transfer a cartridge to disk - loading would in any case be no faster! If you are moving from a disk program to a games cartridge, take out the diskette(s) and TURN OFF THE DRAGON before unplugging DELTA. Plug in the games cartridge before powering on. Failure to do this could result in a dead DRAGON/DELTA/GAMES CARTRIDGE - all three if you are really careless!

DELTA - Quick Syntax Reference

WORD	**SYNTAX**
LOAD	LOAD"filename"
SAVE	SAVE"filename"
RUN	RUN"filename"
	RUN"filename" line number (from keyboard only)
CHAIN	CHAIN"filename"
APPEND	APPEND"filename"
LOADM	LOADM"filename"
	LOADM"filename", address
SAVEM	SAVEM"filename", start address, end address
RUNM	RUNM"filename"
	RUNM"filename", load address
DIR	DIR
	DIR drive
INIT	INIT
CONFIG	CONFIG, drive, tracks, sectors/track, sides, step rate, data rate
KILL	KILL"filename"
ASSIGN	ASSIGN"filename", "newfile"
	ASSIGN"filename",S
	ASSIGN"filename",D
VERIFY	VERIFY D (disable)
	VERIFY E (enable)
SELECT	SELECT drive
COPY	COPY"A:SOURCE", "B:TARGET"
BACKUP	BACKUP
CREATE	CREATE"filename", sectors
FLUSH	FLUSH"filename"
	FLUSH#channel number
OPEN	OPEN#channel, "filename"
CLOSE	CLOSE
	CLOSE#channel
FILES	FILESnumber of channels
PRINT#	PRINT#channel, variable (serial access)
	PRINT#channel, %record number, variable (random access)
INPUT#	INPUT#channel, variable (serial)
	INPUT#channel, %record number, variable (random)
FIND	FIND#channel, search string (serial)
	FIND#channel, %record number, search string (random)

END#	END#channel (serial)
	END#channel, %record number (random)
RESTORE#	RESTORE#channel (serial)
	RESTORE#channel, %record number (random)
DIM#	DIM#(channel number, record length)
IF EOF#	IF EOF#(channel) THEN line number
DO	DO"filename"
BUILD	BUILD"filename"
BOOT	BOOT: RUN"filename"

Error Messages

NOT READY	Drive door open or a non-existent drive has been addressed
SEEK	Required track or record cannot be found
RE-READ	Unable to re-read a sector after it has been written - disabled by VERIFY D
CRC	Checksum error on data or record header
READ ONLY	Disk is write protected!
TRACK NUMBER	Attempt made to access invalid track number. Indicative of a corrupted scratch area. Be careful with your PEEKs/POKEs
DRIVE SPEC	Attempt to access invalid drive (A-D)
	Data lost during a disk read/write data transfer
NOT FOUND	Filename requested not in directory
SYNTAX	Invalid DELTA command syntax
FILE PROTECTED	Attempt to delete or write to a protected file
NO SPACE	Insufficient space to save program to that disk OR to load a program into memory from disk
DIRECTORY FULL	No room left in disk directory for further entries
FILE TYPE	Incorrect file-handling command used. E.g. OPENing a BAS type file in error.
DISK TYPE	Attempt made to access a non DELTA disk

EOF	Attempt made to read or write past the end of file
CHANNEL CLOSED	Attempt to CLOSE an already CLOSED channel
CHANNEL	Invalid channel number specified
RECORD	Invalid record number specified
CHANNEL OPEN	Attempt to OPEN an already OPEN file
WORKSPACE EMPTY	Attempt made to save a null program - minimum size is 13 bytes
NAME EXISTS	Attempt to create or assign existing filename
OPERATION ABORTED	BACKUP, COPY, or INIT has been aborted
PARAMETER	Out of range parameter used
ILLEGAL ACCESS	Attempt made to access a protected disk

Delta Memory Map

Below is a list of locations used by DELTA. This map will be of particular interest to users of 6809 code. It also gives information about areas of memory to avoid when writing your own 6809 routines with DELTA present. All addresses given are in hexadecimal.

0000 - 78FF	Some locations in page three used for system constants BASIC workspace and scratchpad area File Buffer#2 and other buffers - these are defined from the current top of memory when FILES is used User available area as required
7900 – 79FF	FILE buffer #1
7A00 – 7AFF	Disk Direct page and system variables/ constants
7B00 – 7BFF	Directory Buffer
7C00 – 7EFF	Directory bitmap and disk control blocks
7F00 – 7FFF	File control blocks and scratchpad
8000 – BFFF	Extended Colour BASIC-in-ROM
C000 – DFFF	DELTA DISK SYSTEM ROM

E000 – FEFF	ENCODER 09 (Optional extra)
FF00 – FFFF	Input/Output devices

Filenames

DELTA filenames maybe up to eight characters in length and there are few restrictions as to the characters which may be used to make up the name. Alphabetic, numeric and even punctuation can be used within a filename. The first character of the filename does NOT have to be alphabetic as with most other systems.

However, it would be unwise to use a filename such as '(&$!?/<%' as remembering it would be quite a feat of memory! In general, use file names which are relevant to the program in question. The use of a numeral as the last one or two characters will enable you to SAVE different generations of your program as you go along and be able to refer back to an earlier version if necessary.

We STRONGLY recommend that you put your chosen filename in a REM statement at the beginning of each program. You can then LIST it and always be sure of using the correct title. The most popular way of losing a program is to save another program that you are working on onto the LAST program filename you used, thus over-writing the last program and destroying it - this is extremely easy to do when you are tired and it's late at night/early in the morning!

To get round the above problem completely, use the following sequence at the start of every program.

```
1 REM NAME OF PROGRAM IS 'SAFETY6'
2 REM WRITTEN BY A CUSTOMER
3 GOTO 10 : REM SKIP NEXT LINE
5 SAVE"SAFETY6" : END: REM SAVE IT THEN STOP
etc.
```

In normal circumstances, your program would start at line 1, pass line two to line three, and then jump to line 10, hut if you wish to SAVE a program to disk, simply type RUN 5 <ENTER> and line 5 will save the program to disk then stop. If you always want the program to RUN after saving, simply leave out the END.

If you get into the habit of typing RUN 5 every hour you will rarely lose a program and NEVER lose more than an hours' work.

File Planning Summary

1. Decide how much file space you are likely to need (1 domain = 256 bytes). Be generous - diskettes arc quite cheap! Having decided a size, use CREATE to make up the file on disk. DON'T put CREATE in the program which is going to save and retrieve data as it will wipe it!
2. Decide whether to use random or sequential files - read through the chapter on each to decide which is best for your purpose.
3. Decide how many channels will be needed and use FILES to create them - put this in the program right at the beginning.
4. OPEN the file and channel you are going to use.
5. Use either PRINT# or INPUT# to retrieve or store data.
6. CLOSE the channel(s) after use.

DELTA Disk Cartridge - pin connections

1 - 33	odd numbers	=0v	To Drive
2	HLD	Head Load	From Drive
4	WRFLT	Write Fault	To Drive
6	SELD	Select D	From Drive
8	INDEX	INDEX	To Drive
10	SELA	Select A	To Drive
12	SELB	Select B	To Drive
14	SELC	Select C	To Drive
16	MOTORON	Motor On	To Drive
18	DIR	Direction	To Drive
20	STEP	Step	To Drive
22	WRDATA	Write Data	To Drive
24	WRGATE	Write Gate	To Drive
26	TRKOO	Track 0	From Drive
28	WRPROT	Write Protect	From Drive
30	DATAMXD	Mixed Data	From Drive
32	SDSEL	Side Select	To Drive
34	TG43	Track >43	To Drive

All interface signals are active low and are buffered/terminated in the DELTA controller.

Write Fault is not used by DELTA.

The Switchable Drives

One of the more popular types of drive now appearing on the market is the switchable drive. This is normally an 80 track drive with modifications done to the stepper motor circuit. These have appeared in all drive sorts and sizes, There are full height drives, ⅔ height drives and half height drives.

The Odd One Out

The ⅔ height drive is an oddity in that the natural progression of drives has been from full height to half height. As in most modern computer applications things are normally done in the ratio or base of two. That is to say double the memory size, double the disk capacity and so on. Therefore it is only natural that if something for the computer is to be reduced in size then ½ would seem like the correct sum. Somehow the ⅔ height crept in somewhere along the line and is an oddity.

It is unfortunate that the above sum cannot be applied to the price of the drive.

Switching Down

The main purpose of the switchable drive is to enable the user of an 80 track system to READ software or diskettes that have been recorded on a 40 track drive. It is NOT recommended that you use the switchable drive to write or make a 40 track diskette for use in a standard 40 track drive.

Track Width

The earlier section on floppy drives gave the different track pitches for both 40 and 80 track drives. From this it is found that the 80 track drive lays down its tracks over the same area as the 40 track drive. Therefore the 80 track drive has a track thickness of exactly half that of the 40 track drive and a correspondingly smaller gap in the Record - Playback head. It is because of the different track widths that the 80 track drive can get inside the track bands of the larger 40 track drive. The reverse is however not true. If you use one of the switchable drives to make a 40 track diskette you will be making it with narrow track widths and the 40 supposed to read but also the intertrack guard band plus part of the next narrow track. See Figure 5 for a diagram of these effects.

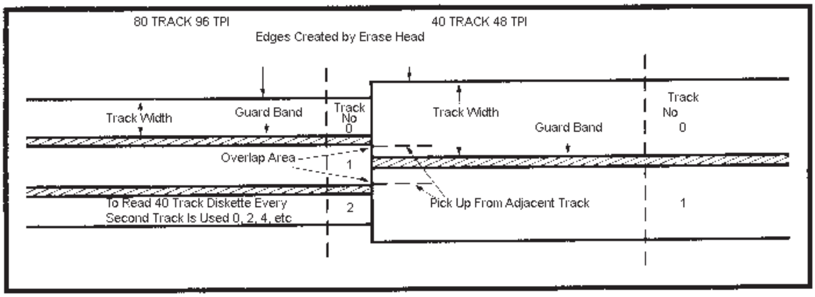

80 TRACK 96 TPI
40 TRACK 48 TPI
Edges Created by Erase Head
Track Width
Guard Band
Track No 0
Overlap Area
1
To Read 40 Track Diskette Every Second Track Is Used 0, 2, 4, etc
2
Track Width
Guard Band
Track No 0
Pick Up From Adjacent Track
1

Figure 5. Here it Call be seen tat the 80 track drive can read inside the 40 track width, but the 40 track read will overlap into the adjacent tracks of the 80 track drive.

Possibilities

If you have no options but to attempt to make a 40 track diskette on a switchable drive then you are more likely to have success if the diskette to be used is first Bulk Erased with a proper magnetic erasure. The other alternative is to use brand new diskettes that have not yet been used or formatted.

Precautions

When using the drives in the 40 track mode be very careful not to run programs that will take the head beyond the natural end track of the diskette. This can be done for example by running the 80 track formatter whilst in the 40 track mode. Extreme damage can be caused to head carriage assembly if this is allowed to happen. At a minimum it is possible to force the drive alignment beyond the specification and the only recourse will be to return the drive for service.

Maintenance

When used correctly then all quality drives will give years of reliable service. This is also true of the switchable drive. There is only minimum maintenance required and that is to keep them clean and after any heavy period of use run a head cleaner diskette. The head cleaning kits available should only be used 'when you feel it is absolutely necessary to do so. Make sure that the diskette is well drenched in cleaning fluid before inserting it into the drive.

DO NOT leave the cleaning diskette rotating in the drive for longer than 15 seconds. That is all the time needed to clean the head(s). Any periods of greater than 15 seconds can result in the diskette drying out sufficiently to become abrasive to the delicate head(s) of the drive. Keep head cleaning to a minimum and certainly not more then once in three months.

Keep Out Rubbish

Always take time to examine your diskettes at regular intervals. Inspection of this kind can often find a potential candidate for the scrap bin before it shows up and destroys some valuable software. In particular look for scored rings or pits on the diskette. Always examine diskettes that have been given to you by another party, before committing them to your drives. Try not to use the drives in a dirty or dusty atmosphere. When possible DO NOT use unbranded diskettes or those of unknown origin. By using branded diskettes you can be sure that the manufacturer has had the confidence to put their own name on the diskette box and label. Simple precautions like this can lead to years of untroubled service from your drives.

Half Height Drives

During the early sections of this book we covered the correct setting up of the DIP or Drive select switches of a typical TEAC Floppy Disk Drive.

With the appearance of the new half height drives it was felt that a new chapter on the setting of some of these drives would be a helpful bonus to those of you who had purchased any of the new type drives. In particular, I will cover the correct setting for the following half height drives. TEAC, TEC and MITSUBISHI.

The drives offered by these companies vary from the standard 40 track single sided drive to the 80 track Double sided drive. In the case of the Mitsubishi drive, it is an 80 Track Double Sided drive.

This is currently the only type of drive made by MITSUBISHI. They do not currently make any other type of drive.

Head Loading

As with the TEAC drives they employ a head load solenoid. This means that it requires either a Motor On or a valid Drive select signal before the head can be loaded against the media for communication with the diskette. This is the purpose of the two switch or link options found on most drives, HM or HS. These stand for Head to Motor and Head to Drive Select respectively. That is to say in the first instance, with the link set to HM, the head of the disk drive will be loaded against the diskette media any time the motor is started up. It may well be that the computer wishes to access another drive, but as all the Motor On lines are usually connected together on the SA400 system, the motors of all the drives on line will be activated. Thus any drive that has the HM option set will load the head of that drive against the media. When the computer comes to use that drive, then the drive will already be in a condition to pass information to and from the diskette.

Head To Select

With the switch or link set in the HS position then the loading of the head against the media will only take place when that particular drive is actually selected for communication between its diskette media and the computer. That is to say that with the HS option selected and a drive set to be drive one or DS1 of the system, then only when drive one is selected will the head of the drive actually be loaded ready for use.

For and Against

There are arguments for and against both methods of head selection. With the HM option there is more wear on the diskette due to the head always being loaded when any drive on the system is selected or used. Any disk drive "LOAD" or "SAVE" command will load all the heads of all the drives on the system whether they will be used or not. Any drive access however small will result in all the heads of all the drives being loaded against the media.

With the head select in the HS position the head of a particular drive will only be loaded when the drive itself is selected for use. That is to say that if drive zero or DS0 was being used, then the head of drive one or DS1 will not be loaded against the media. With the HS option it can be seen that the drive head is only loaded when it is actually required to communicate between the computer and the diskette of that drive.

Settling Time

The arguments go like this. If you have the HM option selected then the drives will not require a Head Settling time before the drive is Ready for use. It speeds up drive to drive access time for certain and is quieter during drive to drive transfers. It also means that the diskettes are subjected to greater wear due to the longer time that they are left in contact with the drive head.

In the case of the HS option then diskette wear is reduced to a minimum as the head is only loaded against the media when an access is required to THAT particular DRIVE. This results in a lot of clicking taking place during drive to drive transfers, as the heads are selected only when they are needed. It also means that when the drive is selected then a small amount of time must be allowed for the head to settle against the diskette before access is attempted to the diskette. This is what is known as Head Settling time and can be as much as 50 m/s in the older type of drive. .

Self Loading

It may be that you have come across a drive that contains neither the HS or HM links or switches. If this is the case then your drive is most likely to be the type whereby the head is loaded against the diskette every time the drive door is closed. This means that the head of the drive is in contact with the diskette at all times. Even when the drive is stationary. This results in an even higher degree of diskette wear than a drive with the HM option set. There is also the likelihood of the diskette being corrupted if the power is removed from the drive whilst the diskette is still inside it. It is one of the penalties to consider when purchasing a drive for any computer. Having said that, it is always recommended that diskettes should NOT be left in any drive when it is not in use. The safest place for a diskette is in the protective sleeve it was supplied in. It is also good practice to only insert the diskette after all the system is turned on and remove the diskette first before any of the system is turned OFF.

Multiplexing

Another connection that can be found on Floppy Drives is the MX link or switch. This is the one that causes most trouble and confusion. Its purpose is to allow two or more drives to be connected together on the same cable. Incorrect setting of this connection can have the most surprising results. It can even cause you to think that there is a fault with another drive on the system.

The Right Way

This drive is unusual in that most drives have this switch working in the opposite sense. That is to say the MX is normally OFF or unmade for multiplexing two or more drives. TEAC have now corrected this anomaly and all the current half height drives will require this link to be unmade for correct operation. It has also been found that this link can be (and normally is) left unmade, even on a single drive system.

Drive Select

With the new type of drives, the drive selection remains the same as with the previous full height drives. If you wish the drive to be the BOOT drive or ZERO drive then you will make the link DS0. If you wish the drive to be drive one then you will make the connection DS1 , Only one of the Drive Select (DSx where x = a value 0 to 3) switches should be made for any drive. No two drives on the same system should have the same Drive Select switches made. When using Double Sided drives with the DELTA Disk System the other side of the diskette is automatically taken care of in the DELTA operating system under the command "CONFIG".

Another Terminator

ALL floppy drives need to have the drive cable terminated with a resistor. Only one resistor should be on the entire system. Normally it is left in the last drive on the cable. These resistors are now appearing in different colours and forms. In most drives they can be identified easily by looking for what looks like an unusually shaped DIL package that is located in a socket. The unwanted terminators can then be unplugged or removed.

Fixed Terminators

With the MITSUBISHI drives a different system is used whereby the terminator is fixed within the P.C.B.. of the drive. With this type of terminator it is necessary to remove a number of links that connect the terminator into circuit. Figure 6 shows a typical link setting for the MITSUBISHI drive. It can be seen that the drive is set to be the BOOT drive or DS0 with the HS option for the head to load only when access is required to that drive. No MX link is made. Whether the drive he used on its on or with another on the same system, this link will be left unmade.

Different Links

Just below these links and to the left is the fixed resistor pack that is the terminator. As it is soldered into the P.C.B., the row of links just below it are all removed to take it out of circuit when required. No other links should be adjusted or tampered with as damage to the drive can result.

Figure 7 shows the correct setting for a TEAC slimline drive. Note the different locations of the links and indeed the different order that they are laid out. The drive is again set to be a BOOT drive or DS0. The white "BECKMAN" resistor pack can be clearly seen. It is this pack which serves as the terminator and is simply unplugged on drives that do not require their use.

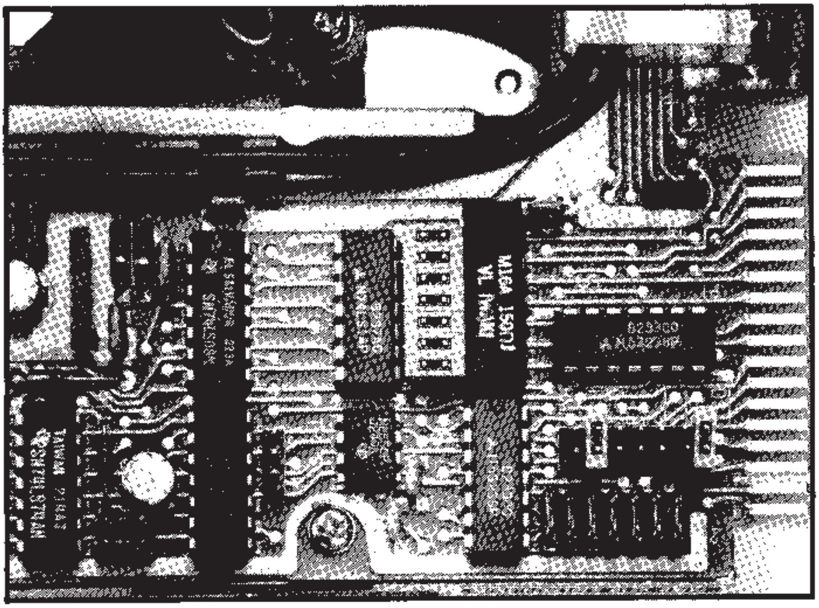


Figure 6. Typical link setting for the Mitsubishi drive.

Rounding Off

In the final example Figure 8 there is nothing wrong with the link settings. Only one link is required for this version or a TEC drive. It does not have a head load solenoid and therefore does not have the usual HM or HS links. The MX link does not require making on this drive therefore only one link is used and that is the one to select the drive. Again the drive is shown as DSO. The terminator or resistor pack is the rather odd looking device spaced to the left of the link bank. It is mounted in a DIL socket and can be removed if not required.

Final Comments

If you are unsure of what to do then always get expert advice. Most manufacturers including CUMANA supply the drives already addressed and with the correct resistor terminator. There is normally no need to open the unit or make any adjustments unless you wish to add other drives to the system, Before making

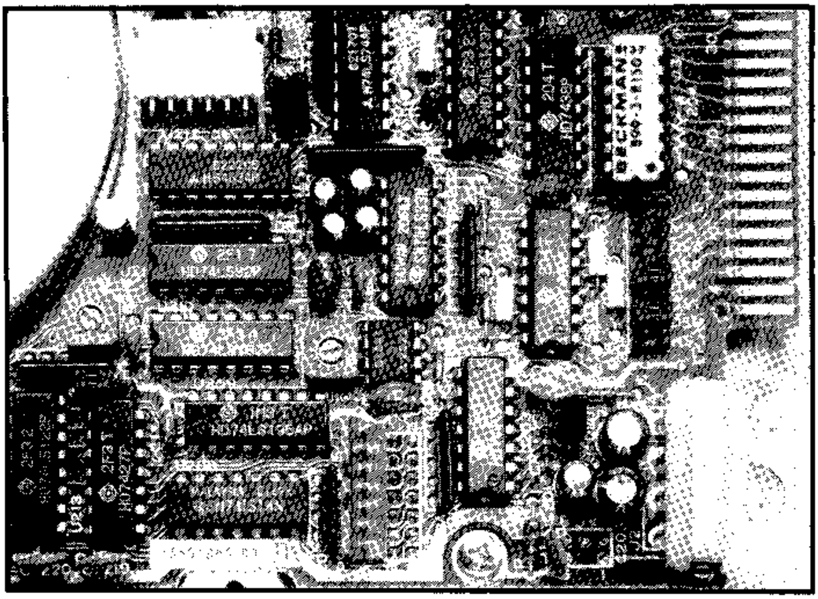


Figure 7. Correct selling for a TEAC slimline drive.

telephone calls for technical advice please read all the literature supplied with your drives, as most answers can be found there.

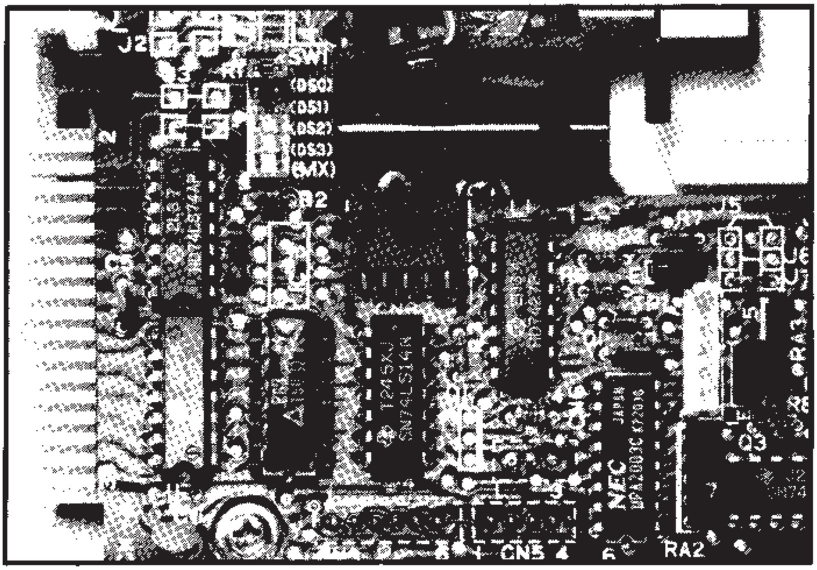


Figure 8. Linking the TEC drive.

Points to Note

Floppy disk storage is very reliable. However, the following points should be noted, especially if this is your first disk system.

1. NEVER switch a disk drive or your DRAGON on or off with a disk inside it - disk corruption is highly likely if you do so.
2. Switch on all your mains powered peripherals (printer etc) before inserting a disk - NEVER turn on a peripheral while a disk is being accessed - open the drive door first.
3. Leave the drive door open when the system is not in use.
4. Disks should be inserted into the drive with the label towards the user and facing the closing trap door - i.e. with the oblong slot facing away from you as you gently push the disk into the drive. See Figure 9.

Care of Diskettes

Floppy diskettes are delicate! Never bend, twist or crease them or subject them to extremes of heat, cold or humidity. Keep them away from food, dust, magnets and smokers!

1. Don't touch the brown recording material held inside the floppy envelope. Grease from fingers can contaminate the head and make the disk impossible to read.
2. When writing on the disk label use light pressure with a felt-tipped pen, NEVER a ball-point or pencil. Wherever possible, write on the label before attaching it to the disk. Place the label in the top right-hand corner of the disk, taking care not to block the write-protect slot (see below). Never stick a label over any of the circular or oblong holes in the disk.
3. Keep all disks in their envelopes when not in use. Avoid leaving them flat on tables. If you have a disk box (supplied free with all 10 off Premier disks), get into the habit of putting the disks straight back in the box after use. It only needs someone to place a cup of coffee on one. . . . . . .
4. Keep young children (and ignorant adults!) away from your disks.

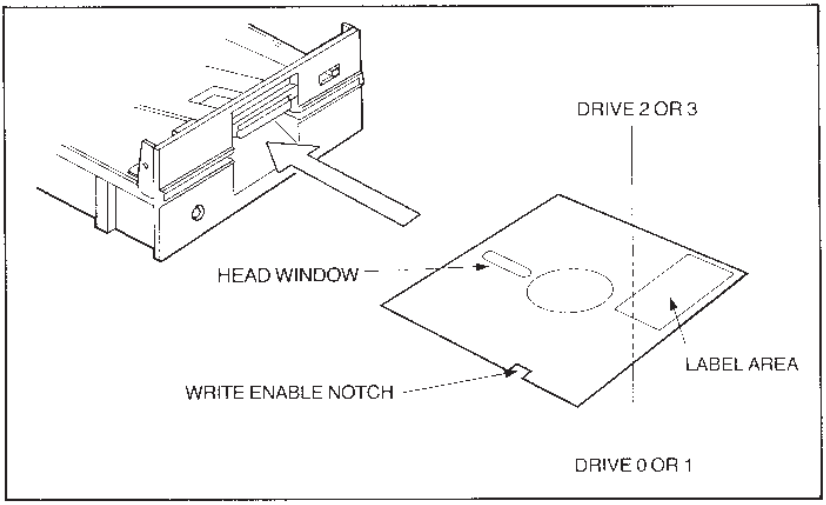


Figure 9. Insertion of the disk.

5. Always make a backup of precious programs on a different disk!
6. Don't leave your disks on top of equipment, especially televisions/VDU's - danger of magnetic contamination.

Write Protection

A disk may be write-protected to prevent any further information being written to it, thus preserving its contents. Part way down one side of the disk is a small square slot. Covering this slot write-protects the disk. Each box of PREMIER disks contains a supply of small white labels for this purpose. They may be carefully peeled off and used again.

Problems - Observations - Help

Problems

If your DELTA system either does not work or works erratically, firstly check all the connections between the DRAGON and the disk drive. Clean the connectors with WD40 or similar aerosol. Check the plugs to make sure you have not inserted incorrectly at some time in the past and bent a contact - spare cables are available from PREMIER and are listed in the price list.

Ensure that all the connectors are pushed gently but fully home. In most instances this will cure your problems.

Have you SELECTed or DIRed a drive not connected to your system? If so type DIR A to return control to your master disk.

If you constantly get read errors from the disk, have you CONFIGed it properly? If you bought a complete system from PREMIER this will not be the problem but if you supplied your own drive you may be attempting to run it too quickly - the drives we supply have a very fast access time. Use CONFIG to alter the step rate to option 3.

Is the diskette you are using faulty? Try another one and see if that stops the errors. Examine the diskette closely for contamination/scoring of the media.

Have you removed the DELTA cartridge with the DRAGON still switched on? If you have. you've probably killed it. Return it to your dealer but note **this fault is NOT covered by the warranty.**

If you are unable, after the above checks, to get the system working, contact your dealer for advice or use our free phone service (see below), Please note however that it is very hard to give an accurate diagnosis over the phone! Don't forget first though – if all else fails read the manual!

N.B. DELTA will not read disks created on or for other computer systems such as BBC, PET, etc. Neither will it read disks created by or for the Tandy Colour Computer.

Observations

If you have any ideas of improvements for the DELTA system or the supplied documentation, please write and let us know. PREMIER always welcome customer feedback - it helps us to produce better products! Address such correspondence to John Hooker.

Help — Customer Phone Service

If you are unable to get DELTA to do something you think it ought to do, PREMIER have a Customer Phone-in Service to help you. It operates from 7-9 p.m. on MONDAY EVENINGS (only) on 01-659-7131. Our engineers are on hand to help you. The service is very popular so please keep trying if the line is engaged.