ACORN COMPUTER

Winchester Disc Filing System USER GUIDE

The Winchester Disc Filing System User Guide

Part no 427000
Issue no 1
Date July 1984

WARNING: THE WINCHESTER DISC UNIT MUST BE EARTHED

Important The wires in the mains lead for the Winchester disc unit are coloured in accordance with the following code:

Green and yellow	**Earth**
Blue	**Neutral**
Brown	**Live**

As the colours of the wires may not correspond with the coloured markings identifying the terminals in your plug, proceed as follows:

The wire which is coloured green and yellow must be connected to the terminal in the plug which is marked by the letter E, or by the safety earth symbol or coloured green, or green and yellow.

The wire which is coloured blue must be connected to the terminal which is marked with the letter N, or coloured black

The wire which is coloured brown must be connected to the terminal which is marked with the letter L, or coloured red.

If the socket outlet available is not suitable for the plug supplied, the plug should be cut off and the appropriate plug fitted and wired as previously noted. The moulded plug which was cut off should be disposed of as it would be a potential shock hazard if it were to be plugged in with the cut off end of the mains cord exposed. The moulded plug must be used with the fuse and fuse carrier firmly in place. The fuse carrier is of the same basic colour* as the coloured insert in the base of the plug. Different manufacturers' plugs and fuse carriers are not interchangeable. In the event of loss of the fuse carrier, the moulded plug MUST NOT be used. Either replace the moulded plug with another conventional plug wired as previously described, or obtain a replacement fuse carrier from an authorised BBC Microcomputer dealer. In the event of the fuse blowing it should be replaced, after clearing any faults, with a 3 amp fuse that is ASTA approved to BS1362.

*Not necessarily the same shade of that colour.

Exposure
Like all electronic equipment, the Winchester disc unit should not be exposed to direct sunlight or moisture for long periods.

Note: Within this publication the term 'BBC' is used as an abbreviation for 'British Broadcasting Corporation'.

The product described in this manual and products for use with it are subject to continuous development and improvement All information of a technical nature and particulars of the product and its use (including the information and particulars in this manual) are given by Acorn Computers in good faith However, it is acknowledged that there may be errors or omissions in this manuaL A list of details of any amendments or revisions to this manual can be obtained upon request from Acorn Computers Technical Enquiries. Acorn Computers welcome comments and suggestions relating to the product and this manual.

All correspondence should be addressed to:
Technical Enquiries
Acorn Computers Limited
Newmarket Road
Cambridge CB5 8PD

All maintenance and service on the product must be carried out by Acorn Computers' authorised dealers. Acorn Computers can accept no liability whatsoever for any loss or damage whatsoever caused by service or maintenance by unauthorised personnel. This manual is intended only to assist the reader in the use of the product, and therefore Acorn Computers shall not be liable for any loss or damage whatsoever arising from the use of any information or particulars in, or any error or omission in, this manual, or any incorrect use of the product.

First published 1984
Published by Acorn Computers Limited Typeset by
Bateman Typesetters, Cambridge

Contents

Introduction

Equipment required

Before you start to use the disc filing system, check that you have all of the following items:

- A Winchester disc drive with a ribbon cable and a power cable.
- Two fixing screws (with washers) for the ribbon cable.
- A guarantee registration card.
- An Advanced Disc Filing System ROM and a BBC Model B Microcomputer.

If any of the necessary items are missing then contact your dealer, quoting the order number given to you when you placed your order. (This number also appears on the dispatch label on the outside of the packing case.)

Text conventions used in this manual

You will notice that the style of printing used to present the text in this manual varies. This is to help you tell the difference between explanatory text, words which appear on your monitor screen (including BASIC keywords) and certain keys on the computer keyboard.

- Ordinary text appears like this, or *like this* for emphasis.
- Text displayed on the screen (including BASIC keywords) appears **like this**.
- Words like **RETURN** mean that you should press the key marked RETURN rather than actually type the letters R E T U R N.

1 What is a disc system?

If you have never used a computer with a disc system before then there are one or two new concepts you need to learn. A disc system consists of some hardware (the disc drive) and some software (a special program called a disc filing system). A disc system enables information to be stored, recovered and organised in a logical way.

Disc drives

As you probably know, your BBC Microcomputer has an internal memory called Random Access Memory or RAM. When you type in a program it is stored in RAM. However, when you switch off the computer everything stored in RAM is lost, so if you need the program again you have to retype it To overcome this problem the computer must be able to transfer the contents of RAM into some form of permanent or 'non-volatile' storage before you switch it off. The User Guide which comes with your BBC Microcomputer describes how to use a cassette recorder to do this. Transferring a program (or text) from RAM to tape is called *saving* it; transferring from tape back to RAM is called *loading* it Although using cassette tape as a storage medium is much better than having to keep typing things in again, it does have some disadvantages:

– The process of saving and loading is quite slow.
– You need to keep track of where on the tape each piece of information is, so that you don't record over it by accident
– You have to wind the tape to the right place yourself.
– Winding from one end of the tape to the other is slow.
– It is very difficult to wind the tape to a particular point accurately.

A disc system has none of these disadvantages; saving and loading information is much faster than using tape, and you don't need to keep a note of exactly where on the disc each bit of information is stored. These 'housekeeping' jobs are done for you by the 'Advanced Disc Filing System', a special program stored in non-volatile Read Only Memory (ROM). The Advanced Disc Filing System is very versatile and is discussed in greater detail later in this chapter.

A disc is a bit like a gramophone record (a disc even has tracks, although you can't see them). A record stores information (usually in the form of music), which is picked up by a stylus as the record rotates on a record player. A disc rotates inside the disc drive, where the information on it is picked up by

something called a read/write head (a bit like the record/playback head on a cassette recorder). Unlike a record, a disc doesn't have to be turned over to get at the information on its other side, and unlike a record, the information on a disc can be erased, changed, or added to.

There are two classes of disc drive, those which drive 'floppy' discs and those which drive ' hard' discs. A floppy disc drive (which you may have heard of) is a small rectangular box with a slot at the front into which the floppy disc is inserted when you want to read information from it or write information to it The Winchester disc drive contains hard discs which are fixed inside the drive itself. More details of the Winchester disc drive are given in the next section.

A disc can hold any information that can be held in your microcomputer memory. The information might be a program, text, or even a computer graphics picture. A piece of information on a disc occupies its own particular area of the disc, called a' file'. Files are not fixed in size, but vary according to the amount of information they contain. A file has a name (decided by you) and an address (decided by the computer), which means that it is easy to get at when you want it Of course, an address by itself may not be much use; it would be no good knowing that someone lived in Acorn Avenue if you didn't know where Acorn Avenue was. You would need a street map, which would enable you to find Acorn Avenue once you had determined which grid square it was in. The computer needs a 'grid' of the disc, which you can tell it to produce through an action known as 'formatting'. Formatting divides a disc into equal partitions known as 'sectors' (see figure 2), and must be carried out before information can be stored on a new, blank disc (the Acorn Winchester disc is delivered already formatted, but this User Guide includes instructions telling you how to reformat the disc should this become necessary). Once a disc has been formatted it stays formatted; you don't have to reformat a disc every time you use it Formatting is fully described in chapter 3.

At this point it is worth noting that your files may be too large to fit into the fixed size of one sector. This is no problem. A file always begins in one sector but may occupy a number of sectors following the first Each sector can hold up to 256 characters or ' bytes'.

The Winchester disc drive and what it does

As we mentioned in the last section, the main difference between the Winchester disc drive and a floppy disc drive is that the Winchester disc is sealed within the disc unit (actually there is more than one disc inside the

Winchester unit). A Winchester disc can hold much more information than a floppy disc of the same size, due to differences in the density of the disc coating and the way in which the read/write head operates. Information can be read and stored much faster with the Winchester because the Winchester disc spins faster, and also spins constantly (until you switch the unit off) which means that it doesn't have to waste time accelerating up to its drive speed before information can be accessed.

When you want to read some of the information on the disc, you give the computer the name of the file containing that information. The computer will move the read/write head to the sector on the disc where the start of the information in the named file is recorded.

The disc filing system

As we have already noted, the main disadvantage of using a cassette recorder to store information is that you need to control the cassette recorder and keep track of the information on it

When using your disc drive all this is done for you by the Advanced Disc Filing System (ADFS). The ADFS is a machine code program produced by Acorn Computers, stored in ROM. Once installed, the program is always there; it is not lost when you switch the computer off. When you **SAVE** one of your BASIC programs the ADFS does the following:

— Finds a free space on the disc big enough for your program.
— Moves the read/write head accurately to the start of the first sector in the free space.
— Transfers a copy of your program from the computer's RAM to the disc.
— Makes a note of where it put your program so as to be able to find it again.

All this is done without you having to think about it and is quite a bit quicker than saving a program on to a cassette tape.

When you save a program you have to give it a name. This is true for the disc system as well as the cassette system. However, the ADFS puts the name to special use. When you type

SAVE "filename" **RETURN**

the ADFS writes the filename, together with the number of the sector on the disc where the file starts, into a ' directory' (also held on the disc). A directory is

like a telephone directory, except that it contains a list of file names and addresses rather than people's names and telephone numbers. When you want the file containing your program back again you simply type

LOAD "filename" **RETURN**

The filing system checks the directory to find out where on the disc to find the file, and then moves the read/write head to that exact place on the disc. The file is then loaded into the computer's memory (RAM) automatically. A number of other facilities are available besides loading and saving programs. These include the ability to copy, delete, rename and restrict access to files. As well as accessing the start of a file, any specific point within a file can be accessed. This `random access' facility is detailed in chapter 6.

Controlling the filing system

The filing system controls the disc drive, and we must be able to give instructions to the filing system. Such an instruction is known as a 'filing system command'. Filing system commands are generally preceded by a * character. They can be typed in directly from the keyboard, in which case they will have an immediate effect, or they can be included in a program. There are also a number of BASIC keywords which have special relevance to files created on a disc; these commands are detailed in chapter 6.

2 Getting started

Before you can use the Winchester disc system you must fit the ADFS ROM to your microcomputer and then connect the drive.

Plugging in the filing system ROM

The instructions for plugging the ADFS ROM into the computer are given in Appendix A. If you have any doubts about fitting the ROM yourself, take your microcomputer and the ROM to your dealer, who will fit it for you.

WARNING: IF YOU FIT THE ROM YOURSELF, REMEMBER THAT ACCIDENTAL DAMAGE CAUSED TO THE ROM OR TO THE MICROCOMPUTER WILL NOT BE COVERED BY GUARANTEE.

Connecting the disc drive

Before you do anything else, make sure that the computer and the drive unit are both disconnected from the mains supply.

Using the broad ribbon cable, connect the 1MHz BUS socket on the drive unit to the 1MHz BUS plug on the underside of the microcomputer (see figure 1).

Note that it is possible (although only just!) to fit the ribbon cable into the computer the wrong way up. The computer end of the ribbon cable will probably have a lug on one surface which fits into a notch at the top of the plug under the computer. If the lug is not there then the correct way to fit the cable socket is with the arrowhead at one end of the socket aligned with the arrowhead next to the 1MHz BUS label on the computer.

Push the plug at the other end of the ribbon cable on to the 1 MHz BUS socket at the back of the Winchester unit. The plug can be secured using the fixing screws provided. Finally, connect the power cable socket to the plug at the back of the Winchester unit, and plug in to the mains supply.

When you have made the connections turn on the power (the disc drive's on/off switch is at the back of the drive unit). The drive unit will start to make a noise a bit like a hair dryer. This is perfectly normal; the sound you can hear is made by a cooling fan and the disc motor.

Starting the filing system

If you have not already done so, switch on the Winchester drive and the computer. The following message (or one very similar) should appear on the screen:

BBC Computer 32K

Acorn ADFS

>_

If 'ADFS' does not appear anywhere in the message (indicating that a ROM other than the ADFS ROM is in the rightmost ROM socket), press **BREAK** while holding down **CTRL** and **A** together. The message will change to that shown above. In either case, wait for about 90 seconds. After this time, the message:

BBC Computer 32K

Acorn ADFS

BASIC

>_

should appear, indicating that ADFS is now ready for use.

If the message

BBC Computer 32K

Acorn ADFS

Drive not ready_

appears, press **BREAK** while holding down **CTRL** and **A** together to restart the initialisation sequence. If the **Drive not ready** message continues to appear, the Winchester disc unit should be returned to your dealer.

If at any time the message

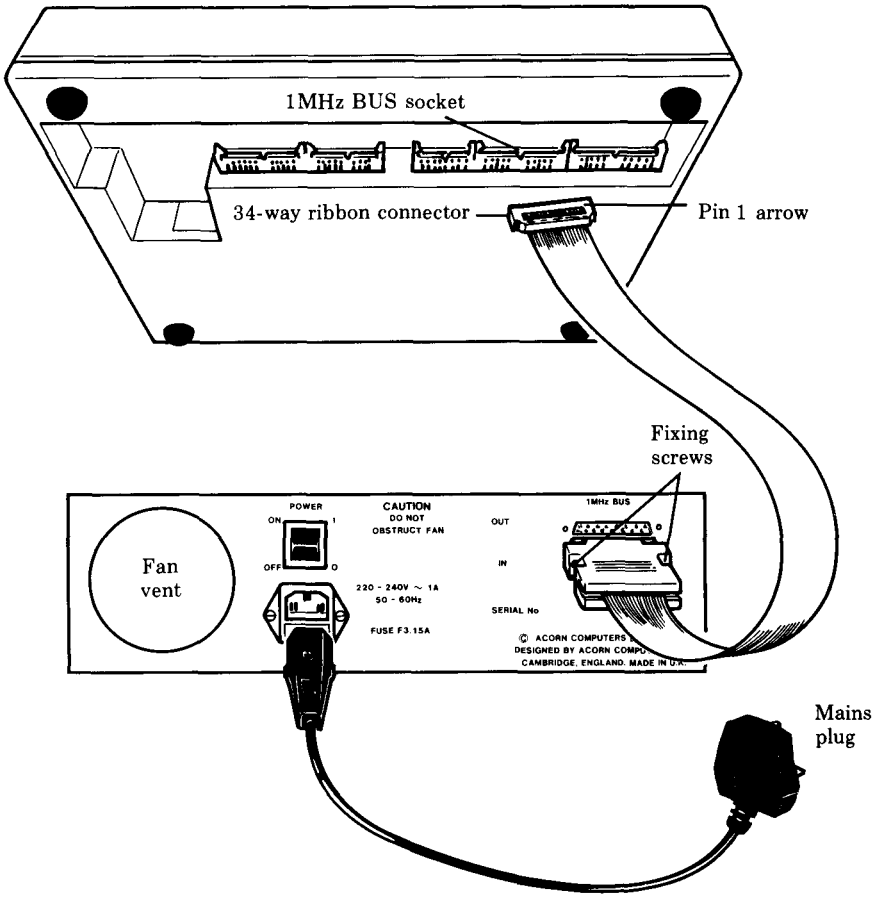
Bad FS map_

appears at the top of the screen, press **BREAK** while holding down **CTRL** and **A** together and continue the initialisation sequence. On rare occasions the error message

Disc error 1C_

may appear. This indicates that the disc is unformatted (or incorrectly formatted), or some kind of hardware failure has occurred. Should this happen, the disc unit must be returned to your dealer.

Figure 1 Connecting the disc drive



3 The Winchester disc

Care of the disc unit

The Winchester disc unit should give good service in normal use. However, you shouldn't bang or vibrate the unit, and it should not be exposed to excessive heat, moisture, direct sunlight or very dusty conditions. The disc unit should be operated standing on its legs on a horizontal surface, and the ventilation slots at the front and the back should be kept clear. The *BYE command (see chapter 5) must be used before the unit is moved.

Storage capacity

Your Winchester disc unit can store about 10.3 megabytes (Mb), where 1 Mb = 1,000,000 bytes or characters. A 10.3Mb Winchester disc unit can store about 5000 pages of text (each page the same size as in this book), so as you can see it is quite a useful storage device. Some of the storage space is reserved for use by the computer itself; this is detailed in chapter 10.

Tracks, sectors and bytes

(see figure 2)

Information is written on to the disc in concentric circles called tracks. Each track is divided into 33 sectors, each of which can contain 256 bytes of information. The 10.3Mb unit has 306 tracks per disc surface, making (4 X 306) = 1224 tracks in total (the unit contains two double-sided discs). The total storage capacity is therefore made up of:

(4 X 306) tracks X 33 sectors X 256 bytes = 10,340,352 bytes

Because (in the above example) the disc unit has four surfaces to store information, instead of having (4 X 306) tracks the unit is said to have 306 `cylinders' (a three-dimensional projection of a circular track would make a cylinder). However, for our purposes we only need to consider one disc surface at a time.

Formatting

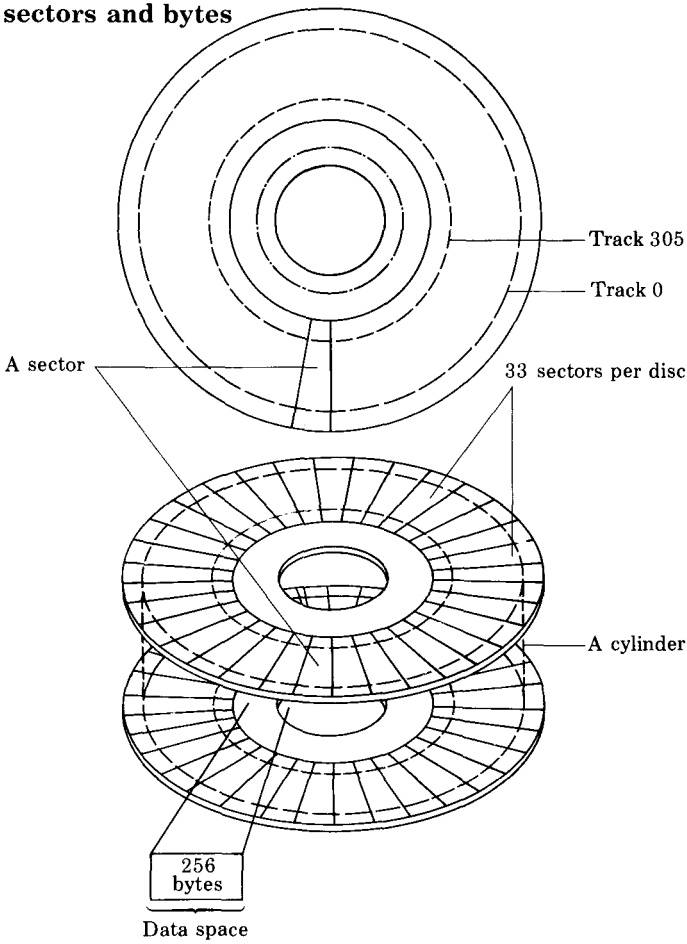
As mentioned previously, before a Winchester disc can be used it must be

formatted. Your Winchester disc has already been formatted and so does not need to be formatted again. Remember, however, that if the

Disc error 1C_

error message is encountered on system start-up (see chapter 2), the disc drive should be returned to your dealer for reformatting. There may be occasions when you wish to delete the entire contents of the disc, enabling you to store new information from scratch. This can be conveniently achieved by reformatting the disc. See chapter 9 (the 'SUPERFORM' utility) for a description of how to do this.

Figure 2 Tracks, sectors and bytes



4 The filing system

Probably the first thing you will want to do with the filing system is to store one of your programs on a disc. You can do this simply by giving the program a name (eg **PROG1)** and then use the **SAVE** command in BASIC, ie

SAVE "PROG1" RETURN

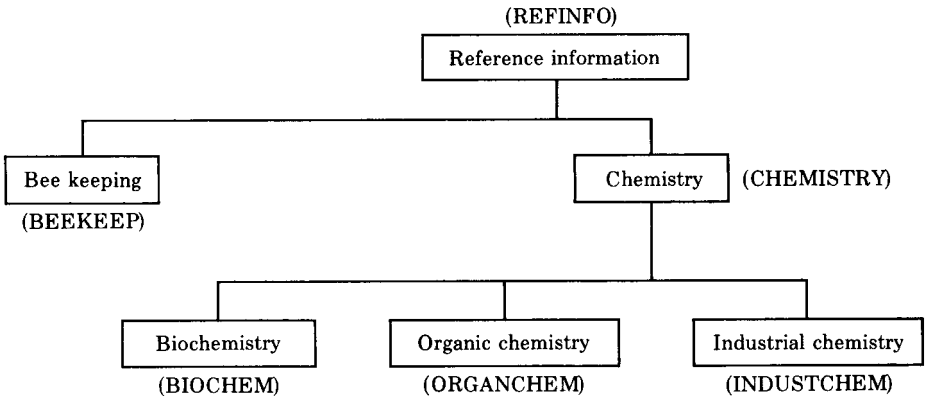
and the filing system does the rest by causing the program to be copied on to the disc. To get the program back again use the **LOAD** command in BASIC, ie

LOAD "PROG1" RETURN

You may be quite happy to type in programs (or text files), give them names, save them in the disc unit, and load them back into the computer when you want them. This is fine so long as you haven't got many files, but as you build up a large collection it will become easy to forget what the contents of each file are and whether they are program files or text files. (It would be a bit like storing books in a library by laying each book out on the library floor, what's more, the books wouldn't have full titles but only abbreviated ones, since a filename can only be up to ten characters long.) Of course, you could keep a separate record of what is in each file, but the filing system is designed to make this unnecessary.

The main feature of the disc filing system is that it is *hierarchical.* The bottom of the hierarchy is a file. The next step up in the hierarchy is a collection of files, known as a ' directory'. Directories can be very useful since they enable files to be collected together into logical groups. The directory also has a name, so it is no longer so inconvenient that the files in the directory only have names up to ten characters long. Going back to the library example, putting files into directories would be equivalent to putting books on to shelves, with each shelf or group of shelves having a name. If we had books about bee keeping then we could probably get them all on one shelf. Books about chemistry would probably need more than one shelf, one for industrial chemistry, one for biochemistry, one for organic chemistry and so on (see figure 3). We see that although the library is organised into different sections for different subjects, some subjects need only one shelf whereas others need more than one.

Figure 3 A typical hierarchical filing system



Now imagine that instead of storing information in books in a library we're storing the same information in files on a disc; we might organise the information as shown in figure 3. The files containing the information are collected together in directories, the directory names (eg BIOCHEM) being shown in brackets. Notice that just as in the library the books about chemistry are on different shelves in the 'master' chemistry section, the information on the disc about chemistry has to be stored in different directories within the 'master' chemistry directory. The directory CHEMISTRY contains not the information itself, but the addresses of the other directories (BIOCHEM, ORGANCHEM, INDUSTCHEM) which contain the information. This illustrates a key feature of the hierarchical filing system provided by ADFS; directories can contain not just files but other directories, which themselves can contain other directories and so on. The structure can be made up to 127 levels deep', although you will find more than five or six levels difficult to keep track of.

Because the system described above can result in directories containing files, other directories, or both, files and directories are collectively known as `objects' – ie a directory (assuming it is non-empty) always contains 'objects', where an object can be a file or a directory. The directory which contains all the highest level objects is called the 'root directory'. The root directory (see below) is created (initially empty) when the disc is formatted.

Pathnames and object specifications

If you wish to refer to a file called (say) Memol, it may not be enough to type, for example,

LOAD "Memo1" RETURN

since Memol may be inside a directory, which may itself be inside another directory and so on. The full specification for an object is called a 'pathname'. From the root directory, the pathname for file Memol at the bottom right of figure 4 is

LE TTE RS.BIZLE TS. Memol

so to load Memol you would have to type

LOAD "LETTERS.BIZLETS.Memo1" RETURN

(note that each part of the pathname is separated by a dot). The above pathname is also called 'object specification', since it specifies how to get to the object (a file in this case) in question.

Directories

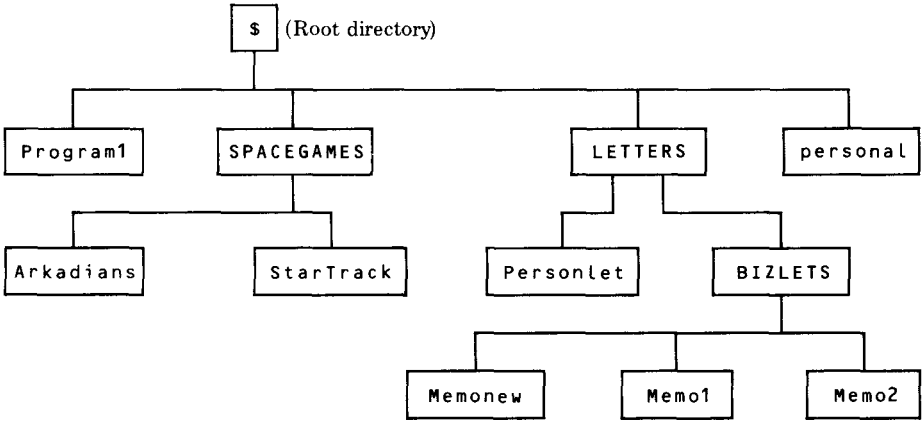
A directory is a collection of objects (up to 47). It can be part of another directory and can itself contain other directories. A directory name can be up to ten characters long, any characters can be used except

 # * . : $ & @ ^

(The above statement is also true of filenames.) The . character is reserved for use in pathnames; the others have special uses which are explained later. Directories are used to divide up other directories into mutually exclusive areas which may contain identical filenames; although the filenames are the same, **A.B.MYPROG** is not the same as **A.B1.MYPROG** because they are in different directories.

Note: Although it is safe to regard a directory as actually containing its constituent objects, what it really contains is a list of disc addresses (plus other information) relating to its objects; see chapter 11 for further information.

Figure 4 The filing system structure



The root directory and the currently selected directory

The root directory is the 'master' directory that contains all the other directories (it may also contain files). The $ symbol (or the & symbol) is used to refer to the root directory (you can't change this). It is created (empty) when the disc is first formatted, and is accessed whenever the disc filing system is first entered. At this point the root directory is said to be the 'currently selected' directory (the C SD). Refer to figure 4; if you asked the computer for a list of all the objects in the CSD (ie $), the list would consist of:

Program1
SPACEGAMES
LETTERS
personal

(**SPACEGAMES** and **LETTERS** are directories, **Program1** and **personal** are files.) If you made **LETTERS** the CSD, and asked for a list of all the objects in the CSD, the list would consist of:

Personlet
BIZLETS

(Here **Personlet** is a file, **BIZLETS** is a directory.)

If you wish to refer to an object in the root directory from another directory then the object's pathname must begin with $; if you are already 'in' the root directory (ie if the root directory is the CSD) then the $ can be omitted (see below for examples).

Object referencing

This section gives a few examples of how objects must be referred to within the hierarchical disc filing system. Refer to figure 4, and assume for the moment that the ADFS has just been entered, ie the root directory (directory $) is the currently selected directory (CSD). Note that in figure 4, directory names are shown all upper case in order to distinguish them from filenames; this convention is adopted here for the purposes of illustration only, and will not be used in the rest of this manuaL File **Program1** would be loaded by typing

LOAD "Program1" RETURN

File **StarTrack** would be loaded by typing

LOAD "SPACEGAMES. StarTrack" RETURN

If you wanted some information about file **Memo1** you would need to type

***INFO LETTERS.BIZLETS.Memo1** RETURN

(the ***INFO** command is detailed in chapter 5).

Now suppose directory **LETTERS** is selected as the CSD (this is done by using the ***DIR** command — see chapter 5). Information about file **Memo1** would now be provided by

***INFO BIZLETS.Memo1** RETURN

and for information about file **Personlet**, only

***INFO Personlet** RETURN

would be necessary. However, if you now wished to load file **StarTrack** you would need to type

LOAD "$. SPACEGAMES. StarTrack" RETURN

since **StarTrack** is not in the CSD. Typing
LOAD "SPACEGAMES. StarTrack" RETURN
would result in

Not found

being displayed as an error message; the computer would be looking for a directory called **SPACEGAMES** within directory **LETTERS**, whereas of course directory **SPACEGAMES** exists only in the root directory. The above instruction would be correct if the root directory were to be reselected as the CSD.

Special characters

^ means 'parent directory', ie the directory of which the CSD is a member. For example, if **BIZLETS** is the CSD then

***INFO ^. Personlet RETURN**

could be typed instead of

***INFO $.LETTERS.Personlet RETURN**

to provide information about file Persontet.

@ means' currently selected directory'. If **BIZLETS** were the CSD, then copying file **Program1** into **BIZLETS** could be achieved by typing

***COPY $.Program1 @ RETURN**

The library directory

The disc filing system enables you to specify one directory as the 'library'. The default library is set up as follows:

— If ADFS is entered with **CTRL BREAK** (or **A CTRL BREAK)** and there is a directory in $ whose name begins with **LIB**, that directory is allocated as the library.

— If ADFS is entered as above, but there is not a directory whose name begins **$.LIB**, **$** (ie the root directory) is allocated as the library.
— If a drive is accessed by the ***MOUNT** command (see chapter 5), the library is not allocated at all (it is said to be 'unset') and has to be set using the ***LIB** filing system command (see chapter 5).

The library can be set to be another directory by using the ***LIB** command, where it will stay until the computer is next switched off or reset by pressing **CTRL BREAK** (or **A CTRL BREAK),** or reset by another ***LIB** command. Any machine code utility programs you use should be located in the library. This is because when you type

* (utility name) **RETURN**

it is equivalent to typing

***RUN** (utility name) **RETURN**

where the directory containing the utility is assumed to be the currently selected directory or the library. The filing system will first search the CSD for the file and then, if it cannot find it there, it searches the library. For example, if directory **$.LIB1** contains a machine code program **MC1** , then typing

***MC1 RETURN**

will run the program whether or not the library is the CSD and whether or not (for systems containing more then one drive) the library is on the currently selected drive.

Wildcard facilities

A means of abbreviating long object names or referring to many objects at once is provided by the ' wildcard' facilities. The filing system commands which can operate with wildcards are followed by the abbreviation <*obspec*> (meaning `wildcard object specification') instead of <obspec> (object specification). ***CAT** is an example of such a command. It provides information about the contents of a named directory. For example, assuming the root directory is the currently selected directory,

***CAT LETTERS.BIZLETS RETURN**

will display information about the directory named **BIZLETS** in the directory named **LETTERS**.

To save having to type out **LETTERS.BIZLETS** the wildcard characters * and # can be used in the object specification. The * character can be a substitute for a string of up to ten arbitrary characters whereas the # character is a substitute for just one arbitrary character. The substitution applies to the first object found, sorted alphabetically. Since **LETTERS** is the only directory in the root beginning with **L** then,

***CAT L*.BIZLETS RETURN**

would do instead of the command given above. In fact, since **BIZLETS** is the only directory in **LETTERS**,

***CAT L*.* RETURN**

would do. The # character as part of an <*obspec*> could be used as follows: if you wanted catalogue information of the directory in **LETTERS**, but you'd forgotten whether it was called **BIZLETS** or **BIZLETZ** then,

***CAT LETTERS.BIZLET# RETURN**

would produce the information without you having to type in

***CAT LETTERS RETURN**

first.

Multi-object operations

Some of the filing system commands can operate on a number of objects instead of just one. These are all followed by the abbreviation <listspec> (short for list specification). ***INFO** is an example of such a command. It provides information about a named object. For example, if **LETTERS** is the C SD,

***INFO BIZLETS RETURN**

will display information about the object named **BIZLETS**. A <listspec> uses the same wildcard characters as <*obspec*> but with greater versatility. For example

***INFO BIZLETS.* RETURN**

would display information about all three files in **BIZLETS**. If you only wanted information about **Memo1** and **Memo2** then you would only need to type

***INFO BIZLETS.Memo# RETURN**

A slightly quicker way of doing this would be to type

***INFO BIZLETS.##### RETURN**

since ##### in this context is a substitute for 'all names with five characters', or

***INFO B*.*0# RETURN**

where * 0# means 'all names with 0 as the penultimate letter'. A quick way of displaying information about file **Memonew** would be to type

***INFO BIZLETS.*w* RETURN**

since * w* in this context is a substitute for 'all names with a w in them'.

Auto-start facilities

Sometimes it is useful to make a program or a file ***LOAD, *RUN** or ***EXEC** automatically when you enter the filing system. This can be done using a file named **!BOOT**. **!BOOT** is a special filename recognised by the filing system when you enter it by pressing **BREAK** while holding **SHIFT** down. If there is a file of specification

$.!BOOT

the filing system will do one of four things according to the option set up by the ***OPT 4, n** command (see chapter 5).

Resetting the system

The **BREAK** key resets the BBC Microcomputer. However, the disc filing system can preserve some of its status after **BREAK** There are two types of **BREAK** the first is called a' hard break' (sometimes called a 'cold start'), and is

achieved by holding down the **CTRL** key and pressing **BREAK** the second is called a 'soft break' (sometimes called a 'warm start'), and is done by just pressing the **BREAK** key.

If a hard break does not give the ADFS start-up message on the screen (see chapter 2) it means that some other filing system is the default filing system (ie the ADFS ROM is not in the rightmost ROM socket). In this case, ADFS can be entered by pressing **BREAK** while holding **CTRL** and **A** down together. (Alternatively, ADFS can be ' warm started' simply by pressing **BREAK** while holding **A** down.)

As far as the ADFS is concerned, hard break is the same as switching the computer off and then on again. The currently selected directory is set to $, the library is set as previously described, and any open files are forgotten about Data written to a file which is still open may be lost

When you do a soft break, the ADFS preserves its status, ie the CSD and the library remain the same, and open files remain open.

Important Following a soft break, or having typed ***DISC** (or ***TAPE**) then ***ADFS** to restart the filing system, the filing system must assume that the contents of RAM below the default value of PAGE (&1 D00 for ADFS) have not been corrupted. This means that if you have been using proprietary software which uses all the RAM from &E00 upwards, you must exit the program with a hard break to obtain normal ADFS operation.

Drive numbers, the MFM floppy disc system and future expansions

Your Winchester disc unit is by convention known as drive 0, although it is only important to remember this if you also have one or more floppy disc drives operating under the MFM (Modified Frequency Modulation) system. The Advanced Disc Filing System handles both types of disc drive, although there are slight operating differences. Full details of the MFM floppy disc system operating under ADFS are given in the *Advanced Disc Filing System User Guide.* If you are operating your Winchester drive in conjunction with MFM floppy drives, drive numbers are allocated as follows:

Drive number	**Disc unit**
0 or A	Winchester drive 0
1 or B	Reserved for possible future expansion
2 or C	Reserved for possible future expansion

3 or D	Reserved for possible future expansion
4 or E	MFM floppy drive 0
5 or F	MFM floppy drive 1
6 or G	Reserved for possible future expansion
7 or H	Reserved for possible future expansion

The possible future expansions referred to above are (drives 1, 2 and 3) additional Winchester drives and (drives 6 and 7) third and fourth MFM floppy drives. Another possible future expansion that is allowed for by ADFS is interchangeable Winchester storage media (see ***DISMOUNT** and ***MOUNT** in the next chapter).

If you wish to refer to an object on another drive then its pathname must start with the appropriate drive number preceded by a colon. For example, if your disc storage system consists of a Winchester and a dual floppy drive and the Winchester is the currently selected drive, then typing in

LOAD " :5.MYPROG" RETURN

would load file **MYPROG** from the root directory of drive 5 (ie the bottom floppy drive). Similarly, all filing system object specifications, wildcard object specifications and list specifications can begin with a drive number. For example,

***COPY ":4.MYPROG :0" RETURN**

would copy **MYPROG** in the root directory of drive 4 to the root directory of the Winchester. (See chapter 5 for more details.) If you don't specify a drive number then the computer assumes that you wish to refer to an object in the currently selected drive. Remember that if you only have the Winchester drive then a drive number need not be included; although

LOAD ":0.MYPROG" RETURN

would work, only

LOAD "MYPROG" RETURN

would be necessary (assuming $ is the CSD).

5 The filing system commands

The ADFS is a 16K byte program. BASIC programs are stored on a disc or tape, but the filing system is stored in Read Only Memory (ROM) inside the BBC Microcomputer. The filing system controls the reading and writing of information to and from the disc unit and provides a number of useful facilities for maintaining that information. The following pages describe all the filing system commands. They are words which the filing system program will recognise and act on. They can be typed directly on to the keyboard or embedded within your BASIC program. They are all prefixed with the * character which signals to the computer that a filing system command (or an operating system command) follows, and should all be followed by a **RETURN**. Each command is described under a number of sections; the syntax abbreviations used are as follows:

< obspec>	= object specification
<*obspec*>	= wildcard object specification
<listspec>	= list specification
< dry>	= drive number

If a syntax abbreviation appears in brackets this indicates that its use is optional.

*ACCESS<listspec> (E) (L) (W) (R)

Purpose

To prevent an object from being accidentally deleted or overwritten. An object is said to have 'attributes' which control the ways in which it can be accessed. Possible attributes are:

E— Execute only. The E attribute is used to protect files containing machine code programs. If the E attribute is set, the file cannot be ***LOADed**, all OSFILE calls except call 6 (delete) are prevented, and display of object information by the ***EX** and *INFO commands is prevented. The only commands which affect a file with the E attribute set are:

RUN** <filename>, *<filename>, ***DELETE, *REMOVE, *DESTROY, *ACCESS** (ACCESS** can only be used to set or remove the **L** attribute— see below.) If the E attribute is set, the R and W attributes (see below) cannot be (if R and W are already set, setting E removes them).

L — Lock If the L attribute is set, the object cannot be deleted or overwritten (until it is unlocked — see 'Notes' below). (Applies to files or directories.)

R— Read access. This must be set for reading (including loading) to be allowed. (Applies to files only, but see 'Notes' below.)

w — Write access. This must be set for writing and updating to be allowed. (Applies to files only.)

Examples

ACCESS $.MC1. E

Gives all files in directory **MC1** in the root the 'execute only' attribute.

ACCESS C LWR

Sets all objects in the currently selected directory with names starting with C with read and write access, and locked against deletion.

ACCESS D.*

Sets all objects in the first directory found with name starting with D with no read/write access, and not locked.

***A. :0.file R**

Sets file in the root directory of drive 0 with read only access, not locked.

Description

Sets the attribute string of a list of objects to the attribute string given.

Notes

There is a further attribute— D (Directory). This is set if the object is a directory, and cannot be changed. Attributes R and w have no meaning to a directory, and are ignored when altering a directory.

It is not necessary to specify attributes when an object is first created— the disc filing system does this for you. The attributes allocated (known as default' attributes) are:

For a file — WR
For a directory — DLR

If an object is locked it will be unaffected by the following commands:

***SAVE**
***DELETE**
***DESTROY**
***RENAME**

An attempt to use any of these on a locked file will result in the error message:

Locked

being produced.
Locking does not protect against erasure by reformatting the drive. The 'locked' attribute can be overcome by using the ***ACCESS** command to 'unlock' a file by giving it new attributes, not including 'locked'.

If the R attribute is not set for a file it cannot be read, and an attempt to use the commands:

***LOAD**
***COPY**

would give the

Access violation

error message, as would an attempt to use the **OPENIN** BASIC keyword (or the OSFIND assembly language call to open a file for reading). Similarly, attempts to use the **OPENOUT** keyword (or the OSFIND call to open a file for writing) on a file without the W attribute set would produce the same result

*ADFS

Purpose

To enter the ADFS from another filing system.

Example

If you had loaded program **PROG** from (say) the BBC Microcomputer disc filing system, typing

***ADFS**
SAVE " PROG" RETURN

would save the program on to the disc. (Note that the program could not necessarily be made to run; see '**COPYF** description in chapter 9 for notes on this subject)

*BACK

Purpose

To go back to the previously selected directory (PSD). The directory selected before the last ***DIR** or ***BACK** command becomes current, and the PSD is set to the old CSD. Thus repeated ***BACKs** may be used to swap between two frequently used directories.

Examples

***DIR A**	Select A as CSD
***DIR B**	Select B as CSD, A is PSD
***BACK**	A is CSD again, B is PSD
***BACK**	B is now CSD, A is PSD

Description

Makes the previously selected directory the currently selected directory.

Associated commands

***DIR**
***CDIR**

*BYE

Purpose

This command must be used before moving the disc unit, and is also a useful command to type in at the end of a session on the computer. It closes all open sequential access files and 'ensures' them on to the disc (ie data held in a buffer in the computer RAM is copied to the disc). The command has the same effect as * CLOSE, and also moves the disc read/write heads to a 'shipping zone'.

Associated commands

***CLOSE**

*CAT (<*obspec*>)

Purpose

To display a 'catalogue' of the specified directory on the screen, consisting of directory header' information (see below) and a list of the objects in the directory. If no directory can be found to match the <*obspec*> an error occurs. If no <*obspec*> is supplied, the currently selected directory is catalogued. The objects in the directory are listed in alphabetical order, together with their attributes and sequence numbers (the latter is a number indicating the 'age' of the object; the first object to be created has sequence number 00, etc - see below for further details).

Examples

```
*CAT
Business Letters          (13)
Drive: 0                  Option 00 (Off)
Dir. BusLet               Lib. Library1

File1       WR (08)   File2       WR (09)   Glenn      WR (00)   XDir       DLR(05)
Z-test-4    WR (12)   Z-test-5    LWR(13)
```

The title of the CSD is **Business Letters** (note a directory *title* is distinct from a directory *name* - see ***TITLE**). The number in brackets following it is the master sequence number (MSN) for this directory. When a new object is added to a directory (or when an existing object is modified to make anew' one) the sequence number of the new object is set equal to the MSN, unless an object with sequence number equal to the MSN already exists, in which case the MSN is incremented and the new value allocated to the new object. **Z-test-5** above was the last file to be created (or modified) since its sequence number is equal to the MSN of the directory. The MSN is a decimal number, running from 00 to 99 and then back to 00 again. The sequence number of **File1** as shown above is 08, that of **File2** is 09. File **Glenn** was the first to be created. The CSD is on drive zero and the option set for this drive (see the ***OPT 4** command) is 0. The directory name of the CSD is **BusLet**, and the current library (which is in the root directory) is called **Library1**. **XDir** is a directory and is locked. **Z-test-4** is a file and is also locked. Object names are displayed in alphanumerical order reading across the four columns.

Description

Displays the catalogue of a directory.

Associated commands

***ACCESS**
***DIR**
***EX**
***INFO**
***OPT 4,n**
***TITLE**

*CDIR < obspec>

Purpose

To create a new directory. A new, empty directory is created with the name given in the < obspec>. The name is also allocated as the directory title, and the master sequence number is initialised as 00.

Example

***CDIR NewDir**

Creates a new directory called **NewDir** in the CSD.

```
*CAT NewDir
NewDir                (00)
Drive:0               Option 03 (Exec)
Dir. :0               Lib. Library1
```

This shows that **NewDir** is an empty directory, and the CSD is the root of drive 0.

Description

Creates a new directory.

Associated commands

***CAT**
***DIR**
***EX**
***TITLE**

*CLOSE

Purpose
To close all sequential access files, and ' ensure' them on to the disc (ie data held in a buffer in the computer RAM is copied to the disc). *CLOSE is the same as the CLOSE#0 BASIC keyword.

Example

***CLOSE**

Description
Closes all sequential access files.

Associated commands

BASIC's **CLOSE# 0**

Notes
For more information on the use of this command, see chapter 6.

*COMPACT (< SP> <LP>)

Purpose

To compact the information on the drive and gather the free space on the disc into larger contiguous sections. This improves access speed, and the

Compaction required

or

Map full

error messages are avoided. The area in RAM used to temporarily hold disc information while the compaction is taking place is the current screen memory unless otherwise specified. <SP> and <LP> are both pairs of hexadecimal digits. <SP> is the start page and <LP> is the length in pages of an area of memory to be used by the command. There must be RAM in the specified area for the command to work correctly. Note that this command will corrupt the RAM contents, so if there is a program or some data in memory that you want to keep, **SAVE** it before you use this command.

Examples

***COMPACT 30 50**

Use **MODE 0, 1** or **2** screen memory.

***COMPACT 40 3C**

Use memory from &4000 up to &7C00, the start of the **MODE** 7 screen RAM.

Description

Compacts the drive, defragmenting the free space.

Associated commands

***MAP**
***FREE**

Notes

The command examines each object on the disc, and if there is some free space just before it, the object is copied, using the specified area of memory, into the free space. (This does not necessarily mean that the free space has to be sufficiently large to accommodate the object If, for example, sector 9 was a free space, and sectors 10 and 11 contained data, ***COMPACT** would move the data to sectors 9 and 10, with sector 11 becoming a new free space.) In this way, objects tend to migrate towards sector zero on the disc and free space tends to migrate towards higher disc addresses, so free space gathers together in larger sections.

If both < SP> and <LP> are absent, ie if just

***COMPACT**
is typed in, the current screen memory (or the start of screen memory to &8000) will be used. As a further example, consider the following sequence (***MAP** is a command which lists the free space on the disc). Suppose that typing

***MAP**

gives

```
Address  :   Length
0002A5   :   000005
000486   :   000002
000689   :   000003
00078C   :   000002
000A97   :   000001
000EFA   :   008E4E
```

Typing

***COMPACT**

followed by

***MAP**

would give

```
Address   :   Length
000689    :   000003
0007CF    :   00000A
000EFA    :
008E4E
```

(Information has been shifted to reduce the number of free spaces, and the first free space in the list is at a higher disc address.)

*COPY <listspec> <*obspec*>

Purpose

To copy a list of files into another directory. All the files referred to by the <listspec> are copied into the directory specified in the <*obspec*>. Memory from the start of the user's BASIC program area (ie the default value of **PAGE**) up to the start of screen memory is used, so * COPY is more efficient in **MODE 7.** Any programs or data in user workspace will be lost

Examples

*COPY $.WOOGIE $.BOOGIE

copies file **$.WOOGIE** into directory **$.BOOGIE** (ie **$.BOOGIE** will contain a new file, **$.BOOGIE.WOOGIE**). Note that files can only be copied into a directory which already exists.

COPY $.WORK.TEST S.WORKBACKUP

copies all files in directory $.WORK which start with TEST into directory **$.WORKBACKUP**.

*COPY * Backup1

copies all files (not directories) in the CSD into a directory in the CSD called **Backup1**.

*COPY :0.ROB1 :4.ROB

copies file **ROB1** on the Winchester drive into directory **ROB** on floppy drive 4.

Description

Multiple file copy.

Notes

The contents of user memory will be lost when this command is used. Also the command is much faster if **MODE 7** is selected first

To copy files into the currently selected directory the special character @ may be used. For example,

***COPY ROB.* @**

would copy all files in directory **ROB** into the CSD.

*DELETE <obspec>

Purpose
To delete a single object from the disc. The space occupied by the object becomes free and available for other information Once an object is deleted you cannot get it back again.

Examples

*DELETE HUGO

Removes an object called **HUGO** from the CSD.

*DELETE $.A.File1

Removes **File1** from directory A in the root directory.

Description
Single object deletion

Associated commands

*ACCESS
*DESTROY

Notes
If you attempt deletion of an object which is locked the error message

Locked

occurs. A directory can only be deleted by unlocking it and deleting all of its constituent files first Also, a directory cannot be deleted if it is the CSD, eg if **$.ADDRESSES** is the CSD then

*DELETE $.ADDRESSES

would produce the error message

Can' t delete CSD

whereas

***DIR $**
***DELETE $.ADDRESSES** (or ***DELETE ADDRESSES**)

would be successful (assuming of course that ADDRESSES is unlocked and empty). If in the above example **ADDRESSES** was the library, then the message

Can' t delete library

would be displayed (for further details, see the ***LIB** command). Also, open sequential files cannot be deleted.

*DESTROY <listspec>

Purpose
To remove a number of objects from the disc in a single operation. A list of the objects which would be removed is displayed, followed by the message:

Destroy ?_

If you do want to remove all the objects listed you must type

YES RETURN

Anything else aborts the command with the message:

Aborted

Example

***DESTROY ***

Remove all objects in the CSD.

DESTROY Work.Temp

Remove all objects in directory **Work** which start with **Temp**.

Description
Multiple object deletion.

Associated commands

***ACCESS**
***DELETE**

Notes
The restrictions on what may be ***DELETEd** also apply to all objects referenced by ***DESTROY**. See ***DELETE** for further details.

*DIR (<*obspec*>)

Purpose

To make a named directory the currently selected directory (CSD). The object specified must be a directory, or if no <*obspec*> is supplied the root directory of the current drive is selected. When the system is first started, or after a hard reset, the CSD is the root directory of drive 0. The command is also used to change or reset the currently selected drive.

Example

***DIR Dir1**

Select **Dir1** as the CSD.

***DIR**

Select the root of the current drive as the CSD.

***DIR ***

Select the first directory in the CSD as the new CSD.

***DIR :1**

Change the currently selected directory to the root of drive 1.

***DIR :1.Work**

Change the currently selected drive to drive 1 (with **Work** in the root directory of drive 1 as the C SD).

***DIR A**

Select the parent of the currently selected directory (ie the directory of which the CSD is a member) as the new CSD.

*DISMOUNT (<drv>)

Purpose

To 'ensure' data on to a disc. The command closes all sequential files, and takes appropriate action if the currently selected directory or library is on the affected drive. This command will mainly be used prior to changing an MFM floppy disc, but also provides for the possible future availability of interchangeable Winchester storage media.

Examples

*DISMOUNT

Ensure data on to the currently selected drive.

*DISMOUNT 0

Ensure data on to the Winchester drive.

Description

Ensures data on to the drive.

Associated commands

*MOUNT

Notes

***DISMOUNT** only closes the files on the drive specified by < drv> (or the currently selected drive if <drv> is absent).

If the CSD is on a ***DISMOUNTed** drive, the system is put into a state such that the CSD and the library are 'unset'. The next access to that drive will either produce the

No directory

error message, or **$** will be read from the drive and treated as the CSD,

according to the nature of the access. If the Winchester is the currently selected drive, typing

```
*DISMOUNT
*CAT
```

would produce (say)

```
$                    (19)
Drive:0              Option 00 (Off)
Dir. "Unset"         Lib. "Unset"
LIBPROG   DLR (22)   PROGX    WR (08)     PROGY    WR (19)
```

The CSD is not set, but is treated as being **$** since **$** is the default directory in the ***CAT** command. However, typing

```
*DISMOUNT
LOAD "PROGX"
```

would produce the

```
No directory
```

error message. The CSD and the library can be set using the ***DIR** and ***LIB** commands

*EX (<*obspec*>)

Purpose

To display information about the directory named in the <*obspec*>. It includes details not given by ***CAT** such as the length of a file and its location (both in hexadecimal). The information is displayed in the following order across the screen:

(Directory header information)

Object name	Attributes	Sequence number	Load address	Execution address	Length in bytes	Start sector

Example

***EX $**

might give

$ (19)
Drive: 0 Option 03 (Exec)
Dir. : 0 Lib. :0

!BOOT	***WR* (17)**	**00000000**	**00000000**	**00000029**	**00007E**
FT	**DLR(01)**	**00000C**			
ROB	**DLR(00)**	**000007**			
X	**WR (19)**	**00000000**	**00000000**	**00000580**	
000167					

(If no <*obspec*> is given, the currently selected directory is examined.) Note that if the object is a directory, only the start sector number is displayed (the other quantities have no meaning for a directory). For a file with the E attribute set, ***EX** will only display the attribute string and the generation number. For example if file x above was given the E attribute, ***EX** would only give

X E (1 9)

Description

Displays directory contents information.

Associated commands

***ACCESS**
***CAT**
***DIR**
***INFO**
***OPT 4,n**
***TITLE**

*EXEC (<*obspec*>)

Purpose

This command reads byte by byte all the information in the specified file as if it were being typed in at the keyboard. Instead of typing in the same sequence of commands repeatedly, an EXEC file (a text file) can be created containing these commands. Typing ***EXEC** <*obspec*> will activate the sequence of commands whenever you want them.

Example

***EXEC HELLO**

Takes the contents of the file **HELLO** and reads it one character at a time as if it were being typed in at the keyboard (and acts upon any commands that are generated).

Description

Execute commands in a file as if typed in at the keyboard.

Notes

One useful application of this command is in using ***OPT 4 3** (see later in this chapter) in concert with ***EXEC !BOOT**. If the file **!BOOT** contains the text **CHAIN** "<*obspec*>" where the object is a BASIC program, pressing **BREAK** while holding **SHIFT** down will automatically load and run the program. See chapter 6 for an example of how to create a **!BOOT** file.

If a file's execution address is &FFFFFFFF, typing

*** RUN** <filename> (or just *<filename>)

will ***EXEC** the file rather than `load-and-run' it

***EXEC** without a filename, when included in an EXEC file, will stop the file from executing. For example an EXEC file could include a line such as:

IF <condition> **THEN *EXEC**

*FADFS

Purpose

This command starts the ADFS without accessing the disc (ie no disc information is loaded into RAM). The system is started with the CSD and the library 'unset', as with *DISMOUNT.

Example

```
*FADF
S *CAT

$                       (19)
Drive:0                 Option 00 (Off)
Dir. "Unset"            Lib. "Unset"
LIBPROG   DLR(22)  PROGX     WR (08)     PROGY     WR (19)
```

Associated commands

***ADFS**
***DISMOUNT**

Notes

Pressing **BREAK** while holding **CTRL** and **F** down together has the same effect as the ***FADFS** command.

*FREE

Purpose

To display the amount of free space left on the disc in disc sectors (in hexadecimal) and bytes (in decimal).

Example

```
*FREE

008E5B Sectors =9,329,408 Bytes Free
000EDD Sectors = 978,176 Bytes Used
```

Description

Displays free space left

Associated commands

***COMPACT**
***MAP**

*HELP <keyword>

Purpose

Displays useful information. For the Advanced Disc Filing System the <keyword> is **ADFS** and the information displayed consists of a list of the filing system commands. If just ***HELP** is typed, the system produces a list of currently installed ROMs.

Examples

***HELP**

Advanced DFS 1.00 ADFS

OS 1.20

***HELP ADFS**

displays a list of the ADFS commands (see chapter 12).

Notes

***RUN, *SPOOL, *SAVE, *EXEC, *OPT, *CAT and*LOAD** are not included in these lists because they are machine operating system commands which operate outside the ADFS. ***HELP** is a machine operating system command.

*INFO <listspec>

Purpose

Displays information about a list of objects. The information is the object name, attribute string and generation number, load address, execution address, length and disc sector address, the same as displayed by ***EX.**

Example

INFO TEST

Displays information about all objects in the CSD with names beginning with **TEST**.

INFO ADD1.

Displays information about all objects in directory **ADD1.**

Description

Displays detailed information about a set of objects.

Associated commands

***CAT**
***DIR**
***EX**

*LCAT

Purpose

Catalogues the current library, as in ***CAT**.

Example

```
*LCAT
$                 (96)
Drive:0           Option 03 (Exec)
Dir. WD1          Lib. $

!BOOT   WR (29)   CINEMAS   WR (96)   UTILS   DR (00)   WD1   DR (02)
WD2     DLR(03)
```

Here the current library is the root directory, and the currently selected directory is **WD1**.

Associated commands

***CAT**

*LEX

Purpose

Examines the current library, as in ***EX.**

Example

```
*LEX
$                    (96)
Drive:0              Option 03 (Exec)
Dir. S               Lib. $

!BOOT     WR (29)    00000000  00000000   00000029  000007
CINEMAS   WR (96)    00000000  FFFFFFFF   0000001E  000008
UTILS     DR (00)    000009
WD1       DR (02)    000062
WD2       DLR(03)    000067
```

Here the current library and the currently selected directory are both $.

Associated commands

***EX**

*LIB (<*obspec*>)

Purpose
Sets the library to the specified drive and directory.

Example

***LIB $.A**

sets directory **A** in the root as the library. After this typing

*< filename>

will search directory **A** for the named file, and if it is found the file will be loaded and executed just as if you had typed

***RUN A.**<filename>

(note that the library does not have to be the C SD). In this example, directory A would not be retained as the library following a hard break or a ***MOUNT** (see below for more details).

Description
Defines the directory that is to contain the library.

Associated commands

***DIR**
***LCAT**
***LEX**
***RUN**

Notes
When ADFS is entered the library directory is set to $, unless there is a directory with name beginning **$.LIB,** in which case this latter directory would be allocated as the library. (If there were two or more such directories, they would be ordered alphabetically, and the first one allocated as the library.) A directory will not be retained as the library following a hard break from ADFS

unless its name begins **$.LIB. A *MOUNT** operation defines the library to be 'unset' even if there is a directory on the drive with a name starting with **LIB.**

The library directory can only be deleted by first of all reallocating another directory as the library. For example, typing

***LIB**

would set **$** as the library directory. The 'old' library directory can then be deleted in the usual way.

The library is usually used to contain files which contain machine code programs, eg games or utility programs.

*LOAD <*obspec*> (<address>)

Purpose

Reads a named file from the disc into memory in the computer starting at either a specified start address or the file's own load address.

Examples

*LOAD "LINDA"

Reads the file **LINDA** into memory starting at the load address of the file when it was saved.

*LOAD LINDA 3200

Reads the file **LINDA** into memory starting at location 3200 (hex).

Description

Loads a file into memory.

Associated commands

***SAVE**
***RUN**

Notes

Note that you should not ***LOAD** programs into memory starting at a location below the default value of **PAGE** for ADFS (&1D00).

The quotation marks shown above are optional (they are not treated as part of the filename, but if you do use them then a pair of marks must be present). If the named file is not found, the message

Not found

is produced.

*MAP

Purpose

To display a map of the free space available on the disc. The format is a list of pairs of numbers of the form:

<Sector address> : <Length in sectors>

If there is a large number of entries in the list the free space on the disc is becoming fragmented, and a creation operation (**SAVE, *CDIR, *COPY** etc) may cause the error message

Compaction required

to be displayed. This can be irritating, so you are advised to ***COMPACT** the disc whenever fragmentation occurs. The ADFS can handle up to 80 entries in the ***MAP** list However, if the list gets to be more than 60 or so entries long, you are advised to carry out a ***COMPACT** operation.

Example

```
*MAP
Address  :  Length
000689   :  000003
0007CF   :  00000A
000EFA   :  008E4E
```

Description

Displays the free space map.

Associated commands

***COMPACT**
***FREE**

*MOUNT(<drv>)

Purpose

This command is mainly used with MFM floppy disc drives (see the *Advanced Disc Filing System User Guide* for full details), but it also provides for the possible future expansions detailed in chapter 4.

The command initialises a drive (by forcing the Winchester controller— a circuit board inside the Winchester unit — to do a ' hard reset), reads the free space map, and stores the appropriate addresses in RAM. It is good practice to ***MOUNT** a drive after a disc error has occurred.

Example

***MOUNT 4**

initialise floppy drive 4.

Description

Initialises a drive.

Associated commands

***DISMOUNT**

Notes

Typing

***DIR** :<drv>

produces almost the same effect as typing

***MOUNT** <drv>

the only discernable difference being that ***MOUNT** defines the library to be 'unset', even if there is a directory on the reselected drive with name beginning with LIB.

*OPT 1 (n)

Purpose

This command enables or disables a message system which displays a file's information (the same as ***INFO).** Every time a file on the disc is accessed the information is displayed. (n) can be anything from 1 to 99 to enable the feature. (n) = 0 disables it

Examples

***OPT 1 1** or ***OPT 1,1**

enables the messages;

OPT 1 0** orOPT 1,0**

disables the messages.

Description

Message system to display file information at every access.

Associated commands

***INFO**

Notes

A space or a comma between * 0 P T 1 and its argument (n) is essential

***OPT 0**

resets the ***OPT 1** state to its default, ie *** OPT 0** has the same effect as ***OPT 1,0.**

*OPT 4 (n)

Purpose

Changes the auto-start option. There are four options to choose from, 0, 1, 2 or 3. Each option initiates a different action when you press **SHIFT** and **BREAK** The computer will either ignore or automatically ***LOAD,*RUN,** or ***EXEC** a file called **!BOOT** which must be in the root directory.

Example

***OPT 4 0** does nothing
***OPT 4 1** will ***LOAD** the file **!BOOT**
***OPT 4 2** will ***RUN** the file **!BOOT**
*OPT 4 3 will ***EXEC** the file **!BOOT**

Description

Changes the start-up option of a disc.

Notes

It is essential to include a space or a comma between the command and (n). ***OPT 40** would produce the message:

Bad option

If option 0 is set the **!BOOT** file need not be there. With any other option the message

Not found

is produced if **!BOOT** is not found after a **SHIFT BREAK**

Important
Do not confuse ***OPT 4** with ***OPT 1** or the BASIC keyword **OPT**; they are completely different

*REMOVE < obspec>

Purpose

To delete a single object from the disc. ***REMOVE** is the same as ***DELETE** except that if the object to be removed does not exist, the error message:

Not found

is not displayed. This command is especially useful as part of a program, since the program will not be interrupted even if the named object does not exist

Example

***REMOVE FRED**

Removes an object called **FRED** from the CSD. (Remember that if **FRED** is a directory it must be empty.)

Description

Single object deletion without error reporting.

Associated commands

***ACCESS**
***DELETE**
***DESTROY**

Notes

The command can be used before an **OPENOUT** command (see chapter 6) to ensure that the default number of sectors is assigned to the file, eg:

```
*REMOVE FRED
F%=OPENOUT "FRED"
```

*RENAME <obspec> <obspec>

Purpose
Changes the object name, moving it to another directory if required.

Example

*RENAME SUMS B.MATHS

Assuming the root directory is the CSD, the object **$.SUMS** becomes the object **MATHS** in directory **$.B** (the original object name **$.SUMS** will no longer appear in the CSD).

Description

Renames an object

Notes
When operating on a directory the root specification (ie 5) may not be used in the directory name. A directory cannot be renamed so as to refer to itself, eg

*RENAME A A.B

would produce the

Bad rename

error message. If the object to be renamed does not exist the message

Not found

is displayed.

The new object name must not contain wildcards, nor can an object be renamed on to a different drive. If the object to be renamed is locked, the **RENAME** operation will not take place and the

Locked

error message will be displayed.

*RUN <*obspec*> (<optional parameters>)

Purpose

This command is used to run machine code programs. It loads a file into memory and then jumps to its execution address, unless the execution address is &FFFFFFFF when the file is ***EXEC** ed as a text file.

Example

***RUN PROG**

will cause a machine code program in the file **PROG** to be loaded and executed starting at the execution address of the file.

Description

Load and run a machine code file, or ***EXEC** a file if the execution address is &FFFFFFFF.

Associated commands

***LIB**
***LOAD**
***SAVE**

Notes

This command will not run a BASIC program.

Typing *<*obspec*> or */<*obspec*> is accepted as being ***RUN<** obspec> (clearly, the object in this case must be a file).

Typing *<filename> results in the file being loaded and executed if it is found in the currently selected directory or the library.

The <optional parameters> referred to above are those which are optional to the machine code program whose filename is in the command. If a file's load address is &FFFFFFFF, * **RUN** <filename> will produce the error message

Won't

*SAVE <obspec> <start address> <finish address> (<execute address>) (<reload address>)

*SAVE <obspec> <start address> + <length> (<execute address> (<reload address>))

Purpose

It is important not to confuse this with the BASIC keyword **SAVE**, they are quite different This command takes a copy of a specified section of the computer's memory and writes it on to the disc, putting it into a file of the given name. You will mostly use this command to record your machine code programs.

Examples

***SAVE "PROG" SSSS FFFF EEEE RRRR**
***SAVE "PROG" SSSS +LLLL EEEE**

SSSS = Start address of memory to be saved
FFFF = Finish address of memory to be saved
EEEE = Execution address (see below)
RRRR = Reload address
LLLL = Length of information

Quotes are optional

Notes

RRRR and **EEEE** may be omitted in which case the reload address and the execution address are assumed to be the same as the start address.

If there are already 47 objects in the specified directory the message

Dir full

is displayed. If the specified filename already exists and is locked the message

Locked

is displayed. If the file already exists but is unlocked it is overwritten. If enough space is available, the information is written on to the disc and the filename is entered on to the catalogue in the current directory.

*SPOOL(<obspec>)

Purpose

Opens a file of the specified name on the disc to receive all the information subsequently displayed on the screen. This is a very useful command particularly for producing a text file of one of your BASIC programs (see 'Notes' below).

Example

You can obtain a text file of one of your BASIC programs as follows:

LOAD "MYPROG"

Loads a program from disc into memory.

***SPOOL TEXT**

Opens a file called **TEXT** ready to receive information from the screen.

LIST

Causes the BASIC program to be displayed on the screen and to be written into the file called **TEXT**.

***SPOOL**

Turns off the 'spooling' and closes the file called **TEXT**.

Description

Spools subsequent output to the screen to a named file opened for the purpose. Closes the file when spooling is terminated.

Associated commands

***EXEC**

Notes

BASIC on the BBC Microcomputer is 'tokenised'. This means that program lines which you type in are abbreviated inside the computer's memory and on the disc. A program file will contain these abbreviated 'tokens' rather than your original program text

*TITLE <title>

Purpose

To change the title of the currently selected directory. The title may be up to 19 characters long. All characters to the right of the command (with leading spaces removed) up to a **RETURN** or double quote are copied into the title field of the CSD. Note that a directory title is distinct from a directory name. The title has no meaning to the computer, it is only used to enable the user to give a directory a 'readable' identity. The directory name will be allocated as the directory title until the ***TITLE** command is used to change it.

Example

S

If typing

```
* CAT
```

gives

```
$                    (48)
Drive:0              Option 03 (Exec)
Dir. $               Lib. $

!BOOT      WR (17)   DIR1       DLR(48)
```

then typing

```
*TITLE Root Directory
*CAT
```

would give

```
Root Directory       (48)
Drive:0              Option 03 (Exec)
Dir. $               Lib. $

!BOOT      WR (17)   DIR1       DLR(48)
```

6 File handling using BASIC

General principles

As we mentioned in chapter 1, one of the major advantages of a disc over a cassette tape is that the read/write head can be moved to a specific place on the disc quickly and accurately. Imagine you have a data file on cassette tape consisting of 'Names' and ' Telephone numbers'. To find a specific telephone number the file must be loaded and read from the beginning until the required record is found. If the file is long this will take some time. On the other hand, the disc filing system enables you to move to the required record and just read that one. Clearly this is much quicker.

To make this possible the disc filing system provides a pointer which points to the next character in the file to be read or written. Files accessed in this way are known as ' random access files'. In BASIC the pointer is controlled by the keyword **PTR#**. The other BASIC keywords which are used in connection with disc files are **EXT#** and **EOF#. EXT#** tells you how long a file is, **EOF#** returns a value of **TRUE** (-1) if the end of the file has been reached and **FALSE (0)** if not All the BASIC keywords used to manipulate disc files are explained in the *BBC Microcomputer System User Guide.* They are:

OPENOUT
OPEN IN
OPENUP
PTR#
EXT#
INPUT#
PRINT#
BGET#
CLOSE#
BPUT#
EOF#

To prepare a file to receive data the **OPENOUT** keyword is used. In the *User Guide* the following example is given:

```
330 X = OPENOUT("cinemas")
```

The effect of this line in a BASIC program is as follows:

1. If a file called 'cinemas' exists it is deleted.

2. A file called 'cinemas' is entered into the currently selected directory.

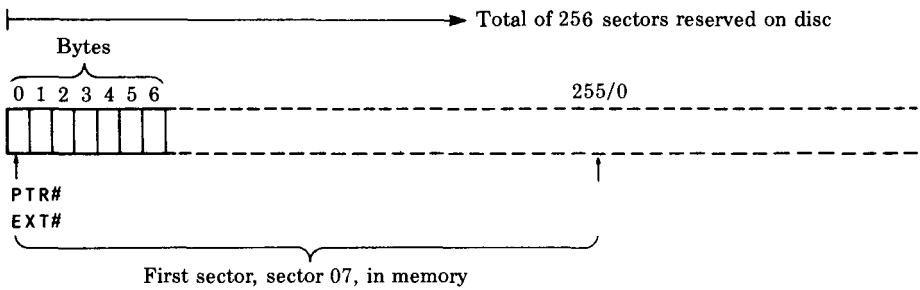
3. The filing system reserves 256 sectors (or the length of the previous file called ' cinemas', if there was one) on the disc for the exclusive use of the file 'cinemas'. If 256 sectors are not available, the file is created using the largest space available. If the disc is full the file is not created and an error message is displayed.

4. Evaluating **PTR#** and **EXT#** at this point will reveal that they are both set to zero.

5. The filing system will have loaded into the computer memory the first sector, 256 bytes, of the file. This area of memory is reserved by the filing system for this purpose and is referred to as the 'buffer'.

Notice that the first action of the keyword **OPENOUT** is to delete any existing file of the specified name.

If there were no files on the disc previously, the effect can be illustrated as follows:

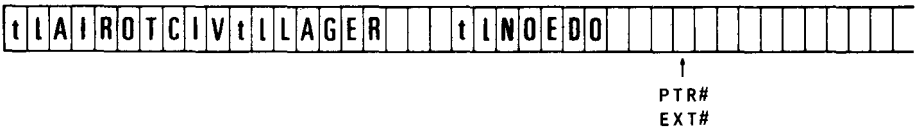


We can now use the BASIC keyword **PRINT#** to write three cinema names into slots of ten characters each, as follows:

```
330 X=OPENOUT ("cinemas")
340 A = PTR#X
350 PRINT# X, "VICTORIA"
360 PTR#X = A+10
370 PRINT# X, "REGAL"
380 PTR# X = A+20
```

390 PRINT# X, "ODEON"
400 PTR# X = A+30

In practice you can do it much more elegantly than shown above; nevertheless the result immediately after line 400 is:



Notice that the cinema names are in the file backwards. They are preceded by two bytes, represented in the diagram by 't' and 'l'. t' specifies the type of data which follows. In this case the type is 'string' so the first byte will contain &00 in hex, as indicated below:

't' = &00 = String type, followed by '1', followed by the string.

't' = &40 = Integer type, followed by four bytes containing the integer.

't' = &FF = Real type, followed by five bytes containing the real number.

In our example the second byte, represented by l', gives the length of the string in hex. The integer and real number types are of fixed length as indicated above so they do not require the byte represented by '1' to give the length Real numbers are stored in exponential format, integers are stored with the high order bytes first in the file.

In the example we have used only the first 26 bytes of the file, so everything written to the file fits into the first sector which is in a 'buffer' in memory. If we had gone on writing names, the filing system would eventually have put the information in the memory buffer on to sector 07 of the disc and loaded sector 08 into the buffer to continue. This is still assuming that there are no other files on the disc, otherwise different sectors would be used. (Remember that sectors 00 to 06 inclusive are put to special uses by the disc filing system— see chapter 10 for more details.) Clearly then, at the end of a sequence of writing actions, we are left with a buffer in memory which may be partly filled with information. We must make sure that this information is written to the disc. This is done with the **CLOSE#** keyword in BASIC (or the ***CLOSE** ADFS command), which empties the buffer and frees the channel on which we opened the file (X in the example).

We can now read the information back from the disc if we want to. **OPENIN** is the BASIC keyword used to do this, eg.

```
5 DIM cine$(3)
10 X = OPENIN ("cinemas")
20 B = 1
30 FOR A = 0 TO 20 STEP 10
40 PTR#X = A
50 INPUT#X,cine$(B)
60 B = B+1
70 NEXT A
```

Line 10 of the example opens the file 'cinemas', loads the first sector into the buffer, sets **PTR#** to zero and **EXT#** to the length of the file.

Lines 30 to 50 of the example read each cinema name into an element of the array **cine$**, advancing the pointer to the start of the next name after reading each one. Now you can see why we stored each name in its own '10 byte record'. This makes it much easier to write a program to find the names again.

The important principle about using random access files is that you must keep track of where each item of information is written. You can then set **PTR#** to point to it again when you want to read or change it The examples illustrate the basis of a very simple technique. There are a number of others which you can devise.

Creating an EXEC file

One useful application of the BASIC disc file handling commands used with ADFS is creating EXEC files. The following program could be used to create a **!BOOT** file; just type in **$.!BOOT** at the Filename? prompt (Press **ESCAPE** to exit the program; don't forget to set up the auto- start option on the disc to ***EXEC** file **!BOOT**.)

```
10 INPUT "Filename",A$
20 F%=OPENOUT A$
30 IF F% ELSE PRINT "Couldn't open" A$: GOTO 10
40 ON ERROR GOTO 130
```

```
  50 REPEAT
  60 INPUT A$
  70 FOR I%=1 to LENA$
  80 BPUT#F%,ASC MID$(A$,I%,1)
  90 NEXT I%
100 BPUT#F%,&0D
110 UNTIL FALSE
120 :
130 IF ERR=17:CLOSE#F%:PRINT "File built":END
140 CLOSE#F%
150 REPORT:PRINT " at "ERL
160 END
```

Once the program has been run, the **!BOOT** file will be ***EXEC** ed by a **SHIFT BREAK** (see ***EXEC** in chapter 5).

Notes

As mentioned earlier, **OPENOUT** reserves 256 sectors for a file. Other files opened may reserve sectors which immediately follow, eg

```
X = OPENOUT ("cinemas")
Y = OPENOUT ("clubs")
```

The statements reserve 512 sectors consecutively (provided the disc was otherwise empty).

It may be that you require more than 256 sectors for the first file 'cinemas'. This presents no problem to ADFS, which, should it become necessary, will move the entire file to a new start address on the disc from where the appropriate number of sectors is available. If this number of sectors is not available anywhere on the disc, the

Compaction required

error message will be displayed.
If you want to create in advance an empty random access file larger than 256 sectors long then for example

```
PTR# OPENOUT "DATA" = &20000
CLOSE#0
```

will create a file 512 sectors (128K) long called **DATA**. You can then open the file later in your program. This method causes the filing system to search the disc for a free space large enough to hold the file. Existing files will be skipped over if they would otherwise overlap with the new file.

Up to ten files may be open at any one time. This is because the space reserved for each file in the computer's memory to hold the information about extent, pointer etc is limited.

7 File handling using assembly language

Chapter 43 of the *BBC Microcomputer System User Guide* is essential reading for anyone wanting to write assembler programs for the BBC Microcomputer. Most of the necessary information for using the filing system in assembly language is presented there. In this chapter the main points are summarised and particular uses of OSWORD are described in detail

General principles

There are a number of operating system routines available to handle disc input/ output All the routines must be called with a JSR and the decimal flag clear. These routines are called in address range &FF00 to &FFFF. Each routine, when called, calls an internal (OS ROM resident) routine whose address is stored in RAM between &0200 and &02FF. (The OS routine is said to 'indirect' through one of these addresses.) The internal routine addresses will vary according to the filing system in use. For example, the routine OSFIND to open or close a file is entered at &FFCE, and is indirected via &021C. &021C and &021D contain the address of (or the 'vector' to) the executable routine in the filing system ROM. (Note that the foregoing is somewhat simplified; a detailed description of the situation for sideways ROMs is beyond the scope of this manual)

Using the available routines you can perform all necessary functions relating to disc files. The relevant routines together with their entry points are summarised below (the third column gives the names of the vectors to the internal routines):

OSFIND	&FFCE	FINDV	&021	C Open or close a file for byte access
OSFILE	&FFDD	FILEV	&0212	Load or save a complete file
OSARGS	&FFDA	ARGSV	&0214	Load or save data about an open file
OSGBPB	&FFD1	GBPBV	&02	A Load or save a number of bytes
OSBGET	&FFD7	BGETV	&0216	Read one byte from an open file
OSBPUT	&FFD4	BPUTV	&0218	Write one byte to an open file
OSWORD	&FFF1	WORDV	&020C	Load or save a number of sectors (and other functions)

OSFIND

Call address &FFCE (indirects through &021 C).

OSFIND is used to open and close files. Opening a file declares a file requiring byte access to the filing system. Closing a file declares that byte access is complete. To use OSARGS, OSBGET, OSBPUT or OSGBPB with a file, it must first be opened.

A=0	Causes a file (or files) to be closed
A=&40	Causes a file to be opened for input only (reading)
A=&80	Causes a file to be opened for output only (writing)
A=&C0	Causes a file to be opened for input and output (random access)

If A=&40 or &C0, the file to be opened must already exist; a new file will not be created.

If A=&80 and the file does not already exist, a new file &10000 bytes (64K) long will be created. If **BPUT#** attempts an access beyond this extent, or if the pointer moves beyond this extent, there may be a delay while new disc space is allocated.

If A=&80 and the file already exists, the disc space used already is allocated to the file; the default 64K can thus be overridden. See OSFILE with A=7.

If A=&40, &80 or &C0, Y (high byte) and X (low byte) must contain the address in memory of the filename, terminated by a carriage return (ASCII &0D). On exit, A will contain the channel number allocated to the file for all future operations (the file ' handle'). If A=0 on exit, then the operating system was unable to open the file.

If A=0 then a file, or all files, will be closed depending on the value of Y. Y=0 will close all files, otherwise the file whose channel number is given in Y will be closed.

On exit from OSFIND, X and Y are preserved, C, N, V and Z are undefined and D=0. The interrupt state is preserved, but interrupts may be enabled during the operation

OSFILE

Call address &FFDD (indirects through &212).

This routine performs actions on whole files. These are loading a file into memory, saving a file from memory, and loading and altering file 'catalogue information' (see A=5 overleaf).

On entry

A indicates the function to be performed. X (low byte) and Y (high byte) point to an 18-byte parameter block, structured as shown below (the left hand column shows addresses relative to the base address given by X and Y).

00 01	Address of filename, which must end with a carriage return	LSB MSB
02 03 04 05	Load address of file	LSB MSB
06 07 08 09	Execution address of file	LSB MSB
0A 0B 0C 0D	Start address of data for save operations or length of file otherwise	LSB MSB
0E 0F 10 11	End address of data to be written (ie byte after last byte) for save operations, or file attributes otherwise	LSB MSB

The following functions are performed by OSFILE according to the value held in A:

A=0 Save a block of memory as a file using the information provided in the parameter block The file's catalogue information (see A=5) will be written into the parameter block.

A=1 Write the named files's catalogue information from the parameter block to the file's entry in the directory.

A=2 Write the named file's load address from the parameter block to the file' s entry in the directory.

A=3 Write the named file's execution address from the parameter block to the file's entry in the directory.

A=4 Write the named file's attributes (see below) from the parameter block to the file's entry in the directory.

A=5 Read a named file's catalogue information (ie load address, execution address, length, type) from the file's entry in the directory. The object type (see below) is returned in A, the other information being written to the parameter block (If the object is a directory, default values are returned for the catalogue information.)

A=6 Delete the named file (the file's catalogue information will be put into the parameter block).

A=7 Create an object This is the same as 'Save' (A=0) except that no data is transferred. This facility can be used to create very large objects for opening for output only, overriding the default length allocation of 64K and avoiding extension delays and possible **Compaction required** errors.

A=&FF Load the named file. The address to which the file is loaded is determined by the least significant byte of the execution address given in the parameter block If this is zero, the address given in the parameter block is used, otherwise the file's own load address is used.

Object attributes are stored in the last four bytes of the parameter block The most significant three bytes are undefined; the least significant eight bits, when set, have the following meanings:

Bit	Meaning
0	The file is readable by you
1	The file is writable by you
2	Undefined
3	The object is locked for you
4	The file is readable by others
5	The file is writable by others
6	Undefined
7	The object is locked for others

In ADFS, bits 4-7 are always identical to bits 0-3. In calls which write the attributes of an object, all bits except 0, 1 and 3 are ignored. If the object is a

directory, bits 0 and 1 are also ignored. Note that 'others' in the above context means other users of, say, the Econet filing system.

Object types returned in the accumulator are:

0	Nothing found
1	File found
2	Directory found
FF	Protected file (E)

On exit

X and Y are preserved, A contains the object type, C, N, V and Z are undefined. Interrupt status is preserved, but may be enabled during a call.

OSARGS

Call address &FFDA (indirects through &0214).

This routine reads or writes an open file's arguments (as defined below), or returns the type of filing system in use, according to the value held in Y on entry. In either case, X (on entry) must point to four bytes in page zero.

If Y is non-zero, then A determines the function to be carried out on the file whose handle is in Y.

A=0	Read file's sequential pointer (BASIC **PTR#**)
A=1	Write file's sequential pointer (BASIC **PTR#**)
A=2	Read file's length (BASIC EXT#)
A=3	Write file's length
A=&FF	'Ensure' the file on to the disc (ie save the latest copy of the file's memory buffer)

(The file length and sequential pointer are sometimes collectively referred to as the file 'arguments'.)

If Y is zero then the following operations are carried out according to the value in A:

A=0	Returns the type of filing system in A:
A=0 —	No filing system currently selected
A=1 —	1200 baud cassette
A=2 —	300 baud cassette
A=3 —	ROM pack filing system
A=4 —	Floppy disc (FM format) filing system

A=5 — Econet filing system
A=6 — Teletext/Prestel Telesoftware filing system
A=7 — IEEE filing system
A=8 — ADFS

A=1 Returns the address of the rest of the command line in the zero page control block

A=&FF Ensures all open files on to the disc.

On exit

X and Y are preserved, C, N, V and Z are undefined, and D=0. The interrupt state is preserved, but interrupts may be enabled during the operation.

OSGBPB

Call address &FFD1 (indirects through &021A).

This routine will transfer a number of bytes to or from an open file, and can also be used to transfer filing system information. If single bytes are transferred using the OSBPUT and OSBGET routines, the 'overhead' incurred for each transfer has a marked effect on performance times. The greater the number of bytes that can be read or written, the more efficient the transfer is; a single OSGBPB call can replace an OSARGS call, and a large number of OSBGET or OSBPUT calls.

On entry

X (low byte) and Y (high byte) point to a control block in memory. A defines the operation to be performed. The control block format is shown below (the left hand column shows addresses relative to the base address given by X and Y).

00	File handle	
01 02 03 04	Pointer to memory area used to transfer data from/to	LSB MSB
05 06 07 08	Number of bytes to transfer	LSB

09 0A 0B 0C	Sequential pointer value to be used for transfer (if used)	LSB MSB

The sequential pointer value given replaces the old sequential pointer value.

The value held in A determines the type of operation:

A=1 Write bytes to disc, using new sequential pointer value

A=2 Write bytes to disc, using old sequential pointer value

A=3 Read bytes from disc, using new sequential pointer value

A=4 Read bytes from disc, using old sequential pointer value

A=5 Read currently selected directory title, boot up option, and drive number. The data returned is:
— Single byte giving the length of the title
— The title in ASCII character values
— Single byte giving the start option
— Single byte giving the drive number

A=6 Read currently selected directory name. The data returned is:
— 1 (length of drive number)
— ASCII-coded drive number
— Single byte giving the length of the name
— The name in ASCII character values

A=7 Read currently selected library name. The data returned is:
— 1 (length of drive number)
— ASCII-coded library drive number
— Single byte giving the length of the name
— The name in ASCII character values

A=8 Read filenames from the CSD. The control block is as follows:

00	CSD master sequence number returned here	
01 02 03 04	Pointer to memory area used to transfer filenames to	LSB MSB

05 06 07 08	Number of filenames to read	LSB MSB
09 0A 0B 0C	File counter (search begins with first file if this is zero)	LSB MSB

The data returned is:

— Length of filename 1
— Filename 1
— Length of filename 2
— Filename 2 .. .

On exit

A requested transfer cannot be completed if the end of the file has been reached, or if there are no more bytes (or filenames) to be transferred. In this case C is set on exit If a transfer has not been completed the number of bytes (or filenames) which have not been transferred is written to the control block (bytes 05-08). The address field (bytes 01-04) is always adjusted to point to the next byte/filename to be transferred, and the sequential pointer always points to the next entry in the file to be transferred.

X, Y and the accumulator are preserved, N, V and Z are undefined, C is set if the transfer could not be completed. The interrupt state is preserved, but may be enabled during the call.

OSBGET

Call address &FFD7 (indirects through &0216). This routine reads a single byte from an open file.

On entry

Y contains the file handle, as set up by OSFIND. The byte is read from the point in the file designated by the sequential pointer, as set up by OSARGS.

On exit

X and Y are preserved, A contains the byte read, N, V and Z are undefined. C is set if the end of the file has been reached, indicating that the byte obtained is invalid. The interrupt state is preserved, but may be enabled during the calL The sequential pointer is incremented.

OSBPUT

Call address &FFD4 (indirects through &0218). This routine writes a single byte to an open file.

On entry

A contains the byte to be written. Y contains the file handle, as set up by OSFIND. The byte is written to the point in the file designated by the sequential pointer, as set up by OSARGS.

On exit

X, Y and A are preserved, C, N, V and Z are undefined. The interrupt state is preserved, but may be enabled during the calL The sequential pointer is incremented, and if this now points to the end of the file, **EXT#** is incremented, and more disc space is reserved as necessary.

OSWORD

Call address &FFF1 (indirects through &020C).

There are four OSWORD calls recognised by the ADFS. They all require X (low byte) and Y (high byte) to point to an area of memory used to contain a control block or for results.

OSWORD with A=&70 – Read the master sequence number and the status byte.

The master sequence number of the currently selected directory is placed in the location pointed to by YX. It is in binary coded decimal form in the range 0-99 inclusive. YX+1 contains a status byte, structured as shown below:

Bit number	Meaning if set
0	File ensuring in progress (IRQ pending)
1	Bad free space map
2	***OPT 1,x** flag – set if messages on

3	(undefined)
4	(undefined)
5	Winchester controller present
6	The Tube is currently in use by ADFS
7	The Tube is present

OSWORD with A=&71 — Read the free space (see *FREE).

The number of bytes of free space on the current drive is written to the memory area pointed to by X and Y. The value given is a 32-bit hexadecimal quantity.

OSWORD with A=&72 — Access the disc controller (reads or writes blocks of bytes to or from the disc).

The control block is shown below.

00	0	
01 02 03 04	Start address in memory of data source or destination	LSB MSB
05 06 07 08 09 0A	Command block to disc controller (see below)	
0B 0C 0D 0E	Data length in bytes	LSB MSB

If the value in byte 00 above is 0, the controller number defaults to 1. As well as the control block shown above, various status bytes in the ADFS workspace are used (eg a byte for the current drive number), and so this OSWORD call will only work if ADFS is the currently selected filing system (the call should not be

made otherwise). If an error of any kind occurs during the execution of the command, the error number will be returned in byte 00 of the control block (0 will be returned otherwise). Error codes are detailed later in this description

The command block is structured as shown below:

Bit

Byte	**7**	**6**	**5**	**4**	**3**	**2**	**1**	0
00	0	0	0	Function code				
01	X	X	X	Disc address				(MSB)
02 03	Disc address							(LSB)
04	Sector count							
05	Unused (set to 0)							

The three bits marked X X X in byte 01 are ORed with the current drive number to give the drive number to use. For a single Winchester drive these bits should all be zero. The function code field is structured as shown below:

Value Meaning

&00 Test drive ready
&01 Restore (ie move read/write head to track 0)
&08 Read
&0A Write
&0B Seek (ie move read/write head to a given disc address)

For functions 0 and 1, disc address (in the command block) should be set to zero. Data length should be set to zero for all functions except 8 and A.

If byte 04 in the command block is non-zero it is used as a sector count, and the data length parameter (bytes 0B—0E of the main control block) is ignored.

For example: to read &1234 bytes starting from sector number &002345 of the current drive, loading into memory at location &FFFF3000 (high bytes FFFF indicating the host machine), the control block would be set up as shown below.

Byte	Value	Meaning
00	&00	Controller number
01	&00	Load address (LS byte)
02	&30	
03	&FF	
04	&FF	Load address (MS byte)
05	&08	Read command
06	&00	Disc address (MS byte)
07	&23	
08	&45	Disc address (LS byte)
09	&00	
0A	&00	
0B	&34	Data length (LS byte)
0C	&12	
0D	&00	
0E	&00	Data length (MS byte)

OSWORD with A=&73 — Read last error information.

This call, if made immediately after a disc error of some kind (including a data error in sequential filing) returns error information (in the control block) as follows:

Byte

00 01 02	Disc address where error occurred, including drive number in three most significant bits of byte 02	(LSB) (MSB)
03	Disc error number, top bit set=valid address	
04	Channel number of file where error occurred	

Only one of bytes 03 and 04 will be valid, depending on the type of error.

8 Changing filing systems

Your computer can have several filing systems available other than ADFS. The following commands are all used to exit from the current filing system into the one named:

***ADFS**	Enters ADFS from one of the others
***DISC**	The floppy disc (FM format) filing system
***DISK**	Alternative spelling for the above
***IEEE**	The IEEE 488 interface filing system
***TAPE3**	300 baud cassette
***TAPE12**	1200 baud cassette
***TAPE**	1200 baud cassette
***NET**	The Econet filing system
***TELESOFT**	The Teletext filing system
***ROM**	The cartridge ROM system

9 The filing system utilities

Your Winchester disc comes complete with a number of 'utility' programs (or just' utilities' for short). The utilities are supplied in directory **UTILS** in the root (although of course you could rename this if you wished). The disc formatter is stored in directory **FORMAT**, also in the root The utilities are put on to the disc when it is formatted prior to delivery. Should your disc prove to be incorrectly formatted, or should you lose your utilities for any other reason, you should go to your dealer who will be able to resupply them.

The utilities are described in detail in the rest of this chapter, and are summarised below.

CATALL

Displays catalogue information (identical to that given by the ***CAT** command) for a whole 'tree', ie the root, all directories in the root, all directories in each of those directories, etc, etc.

EXALL

Similar to CATALL, but displays information similar to that given by the ***EX** command.

WEDITOR

This is a very versatile editor, enabling data on any disc sector to be examined and edited.

COPYF

For copying files between the Winchester, FM floppy disc, and Econet filing sytems.

BACKUP

For 'backing up' data from the Winchester to FM floppy discs, and vice versa

SUPERFORM

Reformats a Winchester disc.

CATALL

Purpose

Displays catalogue information (identical to that given by the ***CAT** command) for a whole 'tree', ie the root, all directories in the root, all directories in each of those directories, etc, etc.

How to use it

CATALL is a BASIC program, so to start it type

CHAIN "$.UTILS.CATALL" RETURN

Notes

The information is scrolled on to the screen, so it is a good idea to put the computer into page mode (or into print mode if you have a printer) before using this utility. The utility can be used from any currently selected directory, but the CSD is automatically reset to the root.

EXALL

Purpose

Displays catalogue information (identical to that given by the ***EX** command) for a whole 'tree', ie the root, all directories in the root, all directories in each of those directories, etc, etc.

How to use it

EXALL is a BASIC program, so to start it type

CHAIN "$. UTILS. EXALL" RETURN

Notes

This command is similar to CATALL, but displays information similar to that given by the ***EX** command.

WEDITOR

Purpose

WEDITOR enables the data in any sector of the disc to be displayed and, if necessary, changed. Data is displayed in 256 byte (ie one sector) blocks, and the data in individual bytes can be accessed, changed, and written back to the disc. WEDITOR is not meant for use as a general purpose editor, you would not, for example, use it to edit one of your BASIC programs. It is mainly intended for editing machine code, editing sequential files, and 'repairing 'damaged' discs by changing data set up by the disc filing system. WEDITOR is a very powerful and versatile tool, but if used wrongly it can be a bit like a chain saw in the hands of a brain surgeon! In particular, you should only attempt disc 'repairs' or modifications if you fully understand the information given in chapter 11 of this User Guide. If you do change the data in a sector, a useful tip is to obtain a printout of the sector contents first, since the original data will otherwise be lost forever once you've changed it

How to use it

WEDITOR is a BASIC program, so to start it type

CHAIN "$.UTILS.WEDITOR" RETURN

A display appears on the screen which represents the contents of a disc sector. The block of zeros on the left hand side of the screen represents a sector split into its 256 bytes (note that there are 16 rows and 16 columns of pairs of digits). This area (which we'll call the ' data area') is used to display the contents of a sector, byte by byte, in hexadecimal form. The block of dots on the right hand side of the screen is called the ' text area', and displays the text equivalent of any hexadecimal ASCII codes which may appear in the data area (this is why you can't use WEDITOR to edit your BASIC programs, as tokenised BASIC cannot be represented in terms of ASCII codes).

WEDITOR has operating instructions 'built in', which are obtained by typing H. Most of the commands are self explanatory, but the use of each of them is illustrated by the following example. Note that although the sector used for the example contains text (the beginning of this chapter in fact), it is shown for illustrative purposes only, you would not normally use WEDITOR to edit a text file. If you'd like to follow through the example using one of your own files, begin by using the ***INFO** command to obtain the file's start sector number (which you should make a note of). Type

R

then type in the sector number (preceded by a & character, since * I N F 0 gives the sector number in hexadecimal), followed by a **RETURN**. The sector used for our example is shown below.

```
Winchester                Disc          Editor                H= Help

39 20 54 68 65 20 66 69 6C 69 6E 67 20 73 79 73    9 The filing sys
74 65 6D 20 75 74 69 6C 69 74 69 65 73 0D D 59     tem utilities..Y
6F 75 72 20 57 69 6E 63 68 65 73 74 65 72 20 64    our Winchester d
69 73 63 20 63 6F 6D 65 73 20 63 6F 6D 70 6C 65    isc comes comple
74 65 20 77 69 74 68 20 61 20 6E 75 6D 62 65 72    to with a number
20 6F 66 20 27 75 74 69 6C 69 74 79 27 20 70 72     of 'utility' pr
6F 67 72 61 6D 73 20 28 6F 72 20 6A 75 73 74 20    ograms (or just
27 75 74 69 6C 69 74 69 65 73 27 20 66 6F 72 20    'utilities' for
73 68 6F 72 74 29 2E 20 54 68 65 20 75 74 69 6C    short). The util
69 74 69 65 73 20 61 72 65 20 73 75 70 70 6C 69    ities are suppli
65 64 20 69 6E 20 64 69 72 65 63 74 6F 72 79 20    ed in directory
55 54 49 4C 53 20 69 6E 20 74 68 65 20 72 6F 6F    UTILS in the roo
74 20 28 61 6C 74 68 6F 75 67 68 20 6F 66 20 63    t (although of c
6F 75 72 73 65 20 79 6F 75 20 63 6F 75 6C 64 20    ourse you could
72 65 6E 61 6D 65 20 74 68 69 73 20 69 66 20 79    rename this if y
6F 75 20 77 69 73 68 65 64 29 2E 20 54 68 65 20    ou wished). The

Current Sector 8124 = 292
```

Note that the number of the sector being displayed is given, in hexadecimal and decimal.

The cursor keys will move the cursor anywhere in the data area or the text area. All keys will overwrite characters in the text area, and keys 0-9 and A—F will overwrite characters in the data area (remember that the data area contains only hexadecimal numbers).

As an example, suppose we wished to change **Your Winchester disc** (near the top of the text area) to **The Winchester drive**. This would be done by positioning the cursor under the **Y** in **Your** and typing in the new text The new text will overwrite the old, and hexadecimal codes for the new text will appear simultaneously in the data area The change could be made the other way round, ie the hex codes in the data area could be overwritten with new ones, and the equivalent alphanumeric characters would appear in the text area

The R command is used to read a sector and display its contents. Simply type in

R

followed by the number (in hex or decimal) of the sector you wish to examine, followed by a RETURN. If you have supplied the correct sector number then type in

Y (or **y**)

in answer to the resulting question Anything other than **Y** or **y** will abort the command. If no sector number is supplied then the sector which is currently being displayed (if any) will be redisplayed.

The **N** and **P** commands cause the next and previous sectors (respectively) to be displayed. For example, typing

N

for the example shown above would cause sector &125 (decimal 293) to be displayed.

The sector display is in fact only a copy (held in a buffei' in the computer's memory) of the sector on the disc. Any changes to the sector will not be implemented until the copy is written back to the disc using the **W** command. If no new address is supplied the 'new' sector will simply overwrite the old one. New addresses should be supplied with care; it is really only safe to write back to where the data came from.

The **S** command is used to set every byte in the buffer to a given value. For example, a sector could be cleared by typing

S followed by **0 RETURN**

The **X** command is used to exchange all bytes of a given value to a new given value. The values can be given in hex or in decimal (both of the same form, or mixed). The two values must be separated by a forward slash, and RETURN must be pressed after the second value has been entered. For example, pressing **X** and typing in appropriate values might give:

Exchange /&70/&71 RETURN

which would change all bytes with value &70 to &71 (of course, the buffer must be written back to the disc if the change is to become permanent).

The **L** command is used to look for a given string. The string can be supplied as hex numbers, decimal numbers (which will be automatically converted to hex), alphanumeric characters, or a string of alphanumeric characters (alphanumeric characters must be typed in in the correct case, but see the **M** command below). All these forms except the last can be supplied as lists. For example, having typed

L

the word 'system' could be typed in as

's
'y
's
't
'e
'm

(pressing **RETURN** after each character) or just as

$system RETURN

Having specified the string, you are asked to specify the limits in the sector within which you wish the search to take place. If you wish the whole sector to be searched, type in

0 RETURN

for the lower limit, and

255 RETURN

for the upper limit

If the specified string is found, the number of the byte which it occupies (or where it starts) is displayed, and an asterisk is displayed just to the left of the relevant byte in the data area. If

Y (or **y**)

is typed in in answer to the **Continue ?** question, the specified string will be searched for elsewhere in the sector, and, if it is not found, in successive sectors. (Any answer other than **Y** or **y** will abort the command.) If you abort the **Look for** command but then change your mind and decide that you'd like to go on looking for the same string, type

G

and WEDITOR will carry on looking, starting at the last sector to be accessed.

The **M** command makes the **L** command more versatile by specifying the 'mask' to be used by WEDITOR when searching for a character. The &DF mask will cause WEDITOR to ignore the case of any character typed in since (for example):

The ASCII code for A is &41 hex = 01000001 binary The
ASCII code for a is &61 hex = 01100001 binary
The only difference in the binary codes is the third bit from the left, which is the only bit not set in the mask (&DF = 11011111 binary).

Notes

Press **ESCAPE** to leave the current command; this also gives an option to leave WEDITOR.

The * key enables any operating system or disc filing system command to be typed in from within WEDITOR (the cursor key must first be positioned in the data area).

Bytes cannot be 'deleted' by WEDITOR, ie a sector cannot be edited down to 255 bytes instead of 256, although of course a byte can always be overwritten with &00 (ie an alphanumeric 'null' character) or &20 (an alphanumeric space). Similarly, bytes cannot be 'added'; a sector cannot be made to contain 257 bytes, nor can bytes be made to 'spill over' into the next sector, although of course any empty bytes can be overwritten with any desired value.

Any hexadecimal numbers which are typed in must be preceded by the '&' character (except when changing values in the data area), and hex letters (ie A, B, C, D, E, F) must always be in upper case.

COPYF

Purpose

Enables files to be transferred between the Winchester, FM format floppy disc, and Econet filing systems. Files can be transferred singly or in predefined groups.

How to use it

COPYF is a BASIC program so to start it type

CHAIN "$.UTILS.COPYF" RETURN

COPYF begins by displaying:

Source filing system :_

This is asking you to specify the filing system from which data is to be transferred. Type

A (for ADFS),

D (for DFS, ie the FM floppy disc filing system), or

N (for Econet)

(Just pressing **RETURN** will cause a list of the available options to be displayed.) The source filing system does not have to be the currently selected filing system. For example, if you are using ADFS and you wish to read some data from a floppy disc (FM format) you don't have to select the floppy DFS first

Having specified the source filing system and pressed **RETURN**, a similar procedure is followed for the destination filing system.

Copying mode :_

is then displayed. Again, a 'help menu' can be displayed by pressing **RETURN.**

If **S** (for **Single file mode**) is typed in then

Source filename :_

is displayed. It is obvious what to do here, but don't forget to type in the full pathname if the file is not in the currently selected directory. Not specifying a filename but just pressing **RETURN** displays a 'help' page — press **SHIFT** to return to the COPYF sequence. Having specified a filename and pressed **RETURN.**

Destination filename :_

is displayed. Again, it is obvious what to do. If you don't supply a filename before pressing **RETURN**, the destination file will be given the same name as the source file. The named file is then copied across and the sequence is repeated. Press **ESCAPE** when you don't want to transfer any more files.

Typing **M** (for **Multiple file mode**) following the **Copying mode:_** prompt will cause

Source list spec :_

to be displayed. Here a list specification is required. For example, to transfer all files in the CSD with names beginning with **FRED**, simply type

FRED*

Having pressed **RETURN**, the files will be transferred one at a time. If you change your mind about transferring a file, simply type in

N RETURN

in answer to the

Copy <filename> (Y/N)

question. Here, the destination filename cannot be specified — it will be the same as the source filename.

Selecting the **List of files** copying mode by typing

L

will result in a sequence of events similar to the multiple file mode, but here the files are transferred one after the other without a break — there is no option to cancel the transfer of a file (apart from pressing **ESCAPE**, which stops the

entire transfer). This mode is especially useful when transferring entire directories — simply type

*** RETURN**

as the list specification.

Notes

In the 'multiple file' or 'list of files' copying modes, the files to be transferred must be in the CSD.

When copying from the Winchester to an FM floppy disc, th

Cat full

error message will be displayed if the disc catalogue is (or becomes) full —COPYF will not select the other side of the disc and continue copying.

If at any stage you wish to abandon a transfer operation, simply press **ESCAPE —** this gets you back to the beginning of the COPYF sequence. If you wish to escape from COPYF completely, press **ESCAPE,** then type

Q

in answer to the

Source filing system :_

prompt

Filing system or operating system commands may be typed in instead of a filename at a 'filename' prompt, but must begin with a space rather than the * character. Note that only filing system commands relating to the 'destination' filing system will be recognised (although of course you can type in a command to select the desired filing system).

If ADFS is the destination filing system and you wish to return to it following a COPYF operation, you must reselect it using the ***ADFS** command.

When using COPYF to transfer programs written under DFS (ie the FM floppy disc system) to the Winchester, the following points should be borne in mind (similar considerations apply when the transfer is the other way):

1. The value of **PAGE** for ADFS (&1D00) is higher than that for DFS and the tape filing system. If a program assigns a value to **PAGE**, you should ensure that it is &1D00 or above. Also, for large programs, the higher value of **PAGE** may mean that the program cannot be loaded into memory.

2. If a program contains instructions which include directory names, an appropriate directory structure must be created on the Winchester — or the appropriate instructions must be changed. For example, suppose we have a program called **A. PROG1,** whose sole instruction consists of

```
10 CHAIN "B. ROG1"
```

For **A.PROG1** to run under ADFS then either.

— Directories **$.A** and **$.A B** must be created on the Winchester, or

- **A.PROG1** could be renamed **PROG1, B.PROG1** could be renamed **PROG2**, and line 10 changed to read

```
10 CHAIN "PROG2"
```

3. Any DFS commands included in a program may need to be changed to their ADFS equivalents (if these exist). For example, a ***DRIVE** command would have to be changed to ***DIR** or ***MOUNT**.

4. Any OSWORD calls with A=&7F will not be recognised by ADFS (see chapter 7 for the OSWORD calls that can be used with ADFS).

BACKUP

Purpose

Enables data to be backed up from the Winchester to FM floppy discs (or vice versa). BACKUP provides several facilities not provided by COPYF, including the ability to back up very large files, ie files too large to be stored on a single floppy disc.

How to use it

BACKUP is a BASIC program, so to start it type

```
CHAIN "$.UTILS.BACKUP" RETURN
```

BACKUP begins by asking you to type in today's date. This is useful as it

enables BACKUP to record the date when a file was backed up, removing the need for you to make a separate note.

Having typed in the date and pressed **RETURN,** a 'menu' is displayed. Before backing up any data, you must type in the floppy drive number that you wish to back up data to (or recover data from). BACKUP begins by assuming that you wish to use floppy drive 0, so if you do there is no need to type **0** in You must now type

B RETURN

or

R RETURN

as appropriate. Note that BACKUP will only recover files from a floppy that have been backed up from the Winchester by BACKUP, so when you use BACKUP for the first time you must use it to back up Winchester files to a floppy. Having typed

R

and pressed RETURN the

Enter file spec :_

prompt appears. The name you now type in must be the full pathname for the file (or files) concerned, ie you must type the name as if **$** is the C SD, even if it isn't; also, the pathname must always begin with **$**, even if **$** is the C SD. You can specify a single file or a list of files (using the * or # wildcard characters). Having entered the file specification, BACKUP asks you to confirm that you wish to back up data to the floppy drive number that you selected at the start of the sequence. This gives you a chance to change your mind, and to check that there is a disc in the selected drive. If you wish to change the drive number for any reason, type

N

followed by the new drive number. If the first drive number is OK, just type

Y

In either case a message will be displayed telling you what is going on and what to do when the backup has finished.

If the extent of the information that you are backing up is sufficient to fill the disc catalogue or even the disc space on one side, a message is displayed asking you to insert a new disc and to specify a new drive number. You need only insert a new disc if you know that there is no more room on the current disc; whether you insert a new disc or not, typing in the new drive number will cause BACKUP to continue. Note that although BACKUP will split a large file into smaller files on either side of a floppy disc (or even on to other discs), when the large file is recovered on to the Winchester it will be reassembled from the smaller files (see `Notes' below for further details).

At this point you may wish to type

E

Or

C

(followed by **RETURN** in each case). **C** simply gives a list of the backed up files; **E** gives a display similar to that given by the ADFS ***EX** command (except that generation numbers and start sector numbers are not given). Both commands also name the Winchester directory from which the files came. The information is automatically displayed in page mode, so **SHIFT** needs to be pressed until all the information has been displayed. You will notice that both commands display a digit to the right of each filename; this will always be zero unless the file concerned is sufficiently large not to be able to fit on to one side of a floppy disc. In this case the digit will be 0 for the part of the file on the first side of the first disc, 1 for the part on the second side, 2 for the part on the first side of the second disc, and so on. The catalogue' function also displays **Start** to indicate part 0 of such a file and **Fragment** to indicate subsequent parts.

Recovering files from a floppy disc back to the Winchester is done by first of all typing

R

You are then asked whether you wish to recover files in **Verbose mode** or **Terse mode.** Verbose mode is assumed unless you type

T

to switch to terse mode. Verbose mode would be used if you only wished to recover certain files from the floppy; each filename is displayed along with with an option as to whether to recover the file or not Type

Y

to recover the file, or

N

to go on to the next file. Terse mode recovers all files from the floppy disc without any option to stop the recovery process (except by pressing **ESCAPE,** but see 'Notes' below). Both modes display the date at which a particular file was backed up. Having selected the recovery mode you want, type in the floppy drive number from which you wish to recover files, followed by the name of the Winchester directory that you wish the files to go to. The latter must be a directory that already exists, and its name need not begin with **$** provided **$** is the Winchester's CSD. Having typed in the directory name and pressed **RETURN** the recovery procedure starts as previously described.

As indicated on the starting menu, operating system or filing system commands (relating to the 'destination' filing system) can be typed in at any **Command/ Drive : _** prompt in the usual way.

Notes

Pressing **ESCAPE** at any time will return you to the start menu. However, pressing **ESCAPE** during the backing up or recovery (terse mode) of a list of files should be avoided as this can result in the

Disc already full

message being displayed the next time you attempt to back up information to the disc in question, whether the disc is full or not

You can back up a file which has the same name as one already on the floppy; the file you back up will not overwrite the existing one. This can be useful if you wish to back up data which varies from day to day, eg weather records; the files containing the weather data could all be called **Weather,** but each would be distinguished by the backup date 'attached' to it by BACKUP.

Discs which have information on them created solely through the use of BACKUP should only be used in conjunction with BACKUP. BACKUP does

not store a file on a floppy disc with exactly the same name as it had on the Winchester, and any file longer than 10K bytes will be split For example, a file 22K bytes long on the Winchester will be split into two 10K files and one 2K file on the floppy. You may wish, for example, to transfer a copy of a program file from the Winchester to a floppy so that you can then take the floppy to another BBC Microcomputer and run the program; in this case it would be advisable to use the COPYF utility. Another point worth noting is that if you delete a backed up file, the BACKUP **E** and **C** functions will show the file as being still on the disc; furthermore, if the disc was full before you deleted the file, an attempt to back up another file to the disc would still result in the

Disc already full

message being displayed. By definition, deleting backed up files is not something you would need to do very often, but if you did want to delete a file the best way would be to delete all the files on the disc (or on one side of the disc, as appropriate) and then back them up again, less the file that you don't require.

SUPERFORM

Purpose

There may be occasions when you wish to clear the disc of all the information held on it; this can be most easily achieved by reformatting the disc, using SUPERFORM. (Remember to copy on to floppy disc any files that you want to keep, especially the utilities!) SUPERFORM is contained in a directory called FORMAT in the root; FORMAT also contains a special file called DEFECTS, whose use is explained below.

How to use it

SUPERFORM is a BASIC program; to start it, carry out a hard break and type

CHAIN "$.FORMAT.SUPERFORM" RETURN

The message:

Defects list not found

may appear. In this case it is still possible to reformat the disc; the 'defects list' and its use are detailed below. The prompt

Action:_

will appear. Pressing **RETURN** at this point causes a 'help menu' to be displayed. At this point you can select the F option without further ado, but you may wish to add to the ' defects list' first The defects list is held in file **DEFECTS** in the same directory as **SUPERFORM**, and is (not unsurprisingly) a list of disc defects of the type which may result in a

Disc error < nn> at : < drv> / < sector number>

error message. The list is compiled when the disc is first formatted prior to delivery, and can be displayed by typing:

T

The defect list is nothing to worry about— ADFS will not let you save data on to a bad area of the disc. The only occasion where you may wish to use the defect list is to add to it when an error message of the type shown above is displayed. The defect list exists as an aid to the formatting process; although reformatting can still take place if the defect list is not updated (or even if the defect list is deleted), a complete defect list will result in increased disc reliability. The defect list can be added to as defects occur, or you can keep a separate list of bad sectors and add them all to the list just before you reformat the disc. Typing

A

gives

Sector ? _

at which point you should type in the number of the bad sector (this can be in decimal or — preceded by the **&** symbol — hexadecimal). Having entered the sector number and pressed **RETURN**, the above question will be repeated. If there are no more additions to the list, simply press **RETURN.** A list of the sector numbers you have just entered will be displayed, followed by a question which asks you to confirm that the list is correct Type

Y

if the list is correct, or

N

if it isn't (in the latter case you will have to re-enter the list). Pressing **Y** will add the new defects to the list and a confirmation message will be displayed followed by the

Action:_

prompt At this point

S

should be typed to save the defects list to the disc. Typing

F

in answer to the

Action:_

prompt will result in

You really want to format drive 0 ? _

being displayed. This is to give you one last chance to change your mind. If you type in anything other than

YES RETURN

the formatting sequence will be aborted (the contents of any files which may have been on the disc are preserved). A correct reply will result in the message

Fill byte (RETURN for default) ? _

being displayed; simply press **RETURN**, which will result in

Formatting..

being displayed. This indicates that the computer is carrying out the formatting process. About 40 seconds later, the message

Verifying..

is displayed, which indicates that the computer is checking the state of the disc. After about two minutes the message

No defects found

will appear. However, if the disc has defects you will get a message (or several messages) such as:

Defect 19 at 001BC3

In either case, as soon as the final prompt line, ie

>_

appears, the disc is ready for use. SUPERFORM will have reformatted the disc, and will also have recreated a directory called FORMAT, saving itself and the defects file into it. (The first thing you should do now is to save the utilities back on to the Winchester!)

Note that the **B** and **C** options are only intended for use by Acorn Computers Limited when the disc is formatted prior to delivery. In particular, the **C** option is only intended for use with non-standard disc drives and should *not* be used.

10 Error messages

This chapter lists all the ADFS error messages, each preceded by the appropriate error code. Error codes are not displayed, but are included here to enable you to include error handling sections in any of your programs which include ADFS commands (the main list is in alphabetical order of error names, but a list in numerical order of error numbers is also given).

&92 Aborted

Something other than **YES** (or **Yes**, or **yes**, etc) has been typed in in response to the question

Destroy?

following a ***DESTROY** command.

&BD Access violation

An attempt has been made to read (or load) a file with the R attribute not set, or to write to a file with the **W** attribute not set.

&C2 Already open

An attempt has been made to delete (or save a new version of) a file which is open. Also occurs if an attempt is made to open a file which is already open (unless both ' opens' are for input only). See ***CLOSE**.

&C4 Already exists

An attempt has been made to create a new object with the same name as an already existing object This includes ***CDIR** and ***RENAME** but not ***SAVE** or BASIC's **SAVE**.

&AA Bad checksum

RAM is corrupted, which prevents ADFS from being able to close a file or read or write to it The system must be restarted by a hard break

&FE Bad command

The command given was not recognised by the ADFS, nor was it found as a utility in the CSD or the current library.

&A9 Bad FS map
Either RAM or disc sectors 0 or 1 are corrupted. The system must be restarted by a hard break

&CC Bad name
An illegal filename was used, ie one including **$** (dollar) or **:** (colon) outside the context of a root specification, or with a zero length component of a pathname, or other special characters in the wrong context, eg

```
*EX $$
*DIR FILE:ONE
*DIR DIR..XDIR1
*EX A@B
*EX A^B
```

&CB Bad opt
An invalid argument has been assigned to a ***OPT** command.

&94 Bad parms
Invalid parameters were given with a ***COMPACT** command to specify the RAM area to be used.

&B0 Bad rename
An attempt has been made to rename a directory in such a way as to produce an illegal directory structure, eg

```
*RENAME A A.B
```

so that directory **A** contains a reference to itself. This is illegaL

&A8 Broken directory
An attempt has been made to access a directory which is in some way corrupt and as such should not be accessed. This error implies that the disc is in an inconsistent state and should be reformatted if possible (return the drive to your dealer if reformatting would mean the loss of the utility programs).

&96 Can't delete CSD
An attempt has been made to delete the currently selected directory.

&97 Can't delete library
An attempt has been made to delete the current library.

&DE Channel on channel <nn>
A sequential file operation has been attempted with an illegal or unassigned file handle. <nn> is decimal

&98 Compaction required
A creation operation (eg **SAVE**, ***CDIR**, ***COPY**) has been attempted on a disc where the free space has become too fragmented.

&CA Data lost on channel <nn>
A disc error of some. description occurred during accessing a sector from the disc during an attempt to read or write an open file. Caused by memory being illegally overwritten (or hardware problems). <nn> is hexadecimaL

&B3 Dir full
An attempt has been made to create a new object in a directory already containing 47 entries.

&B4 Dir not empty
An attempt has been made to delete a directory which still contains objects.

&C7 Disc error <nn> at :<drv>/<sector number>
A fault on the disc was detected by the controller during the last operation. <nn> is the error code, <dry> is the drive number, <sector number> is the sector number (in hexadecimal) where the error was discovered (if appropriate). Some error codes are:

01 — No index. Is the drive formatted?
02 — Seek error
03 — Write fault
11 — Data checksum error
12 — Address mark not found
14, 15 — Seek error
18 — Data checksum error
1A — Format error
20 — Invalid command to controller

21 — Illegal disc address
24 — Invalid parameter to controller
25 — Illegal drive number

(The error codes are hexadecimal values, the same as would be returned by an OSWORD &72 call.)

&C6 Disc full

There is not enough free space on the drive to carry out the requested operation. This includes ***CDIR**, ***SAVE** (and BASIC's **SAVE**), and opening new files or extending existing files.

&CD Drive not ready

The drive is not yet up to speed (may occur if the drive has just been started). If this continues to appear, the disc unit should be returned to your dealer.

&DF EOF on channel <nn>

End of file. This error occurs if two consecutive attempts have been made to read from a file whose end has been reached. The failure of the first attempt will have been flagged by the contents of the C flag following an OSBGET or OSGBPB command (see chapter 7). <nn> is decimal

&C3 Locked

An attempt has been made to remove, rename or overwrite an object which is locked.

&99 Map full

The free space map is full. The disc should be ***COMPACTed**, otherwise it may not be possible to save further information to it

&D6 Not found

The object referred to was not found.

&C1 Not open for update on channel <nn>

An attempt has been made to write to a random access file which is only open for reading. <nn> is decimal

&B7 Outside file on channel <nn>

An attempt has been made to set the pointer of a file which is only open for reading to a value beyond the end of the file. <nn> is decimal

&95 Too many defects
Too many media defects were found during formatting (the disc unit should be returned to your dealer).

&C0 Too many open files
An attempt has been made to open an eleventh file. Only ten files may be open at once.

&FD Wild cards
A wildcard character (***** or **#**) was found where a full object specification is required, eg in ***DELETE, *SAVE, *CDIR.**

&93 Won't
An attempt has been made to ***RUN** a file whose load address is &FFFFFFFF.

Error codes - numerically ordered list

Hex	Decimal	
&92	146	Aborted
&93	147	Won't
&94	148	Bad parms
&95	149	Too many defects
&96	150	Can't delete CSD
&97	151	Can't delete library
&98	152	Compaction required
&99	153	Map full
&A8	168	Broken directory
&A9	169	Bad FS map
&AA	170	Bad checksum
&B0	176	Bad rename
&B3	179	Dir full
&B4	180	Dir not empty
&B7	183	Outside file on channel <nn>
&BD	189	Access violation
&C0	192	Too many open files
&C1	193	Not open for update on channel <nn>
&C2	194	Already open
&C3	195	Locked
&C4	196	Already exists
&C6	198	Disc full
&C7	199	Disc error <nn> at <drv>/<sector number>
&CA	202	Data lost on channel <nn>
&CB	203	Bad opt
&CC	204	Bad name
&CD	205	Drive not ready
&D6	214	Not found
&DE	222	Channel on channel <nn>
&DF	223	EOF on channel <nn>
&FD	253	Wild cards
&FE	254	Bad command

11 Technical information

General

Sectors 0 and 1 on a drive contain the total number of sectors on the drive, the boot option number, and the free sector gap list Sectors 2 to 6 inclusive are the root directory.

The free space map

The free space map (FSM) is stored in sectors 0 and 1 on each drive. The format is:

Sector 0

0	Disc address of first free space (LS byte)
1	Disc address of first free space
2	Disc address of first free space (MS byte)
3	Disc address of second free space (LS byte)
4	Disc address of second free space
5	Disc address of second free space (MS byte)
6	Disc address of third free space (LS byte)
	:
	:
	:

etc for all other free space up to 82 entries

	:
	:
246	Reserved
247	Reserved
248	Reserved
249	Reserved
250	Reserved
251	Reserved
252	Total number of sectors on disc (LS byte)
253	Total number of sectors on disc
254	Total number of sectors on disc (MS byte)
255	Checksum on free space map, sector 0

Sector 1

0	Length of first free space (LS byte)
1	Length of first free space
2	Length of first free space (MS byte)
3	Length of second free space (LS byte)
4	Length of second free space
5	Length of second free space (MS byte)
6	Length of third free space (LS byte)
	:
	:
	:

etc for all other free space up to 82 entries

	:
	:
246	Reserved
247	Reserved
248	Reserved
249	Reserved
250	Reserved
251	Disc identifier
252	Disc identifier
253	Boot option number
254	Pointer to end of free space list
255	Checksum on free space map, sector 1

The disc addresses and lengths are in sectors. The free space map is stored in RAM from &0E00 to &0FFF when ADFS is selected, so the first free space pair is held at &0E00, the second at &0E03, and so on.

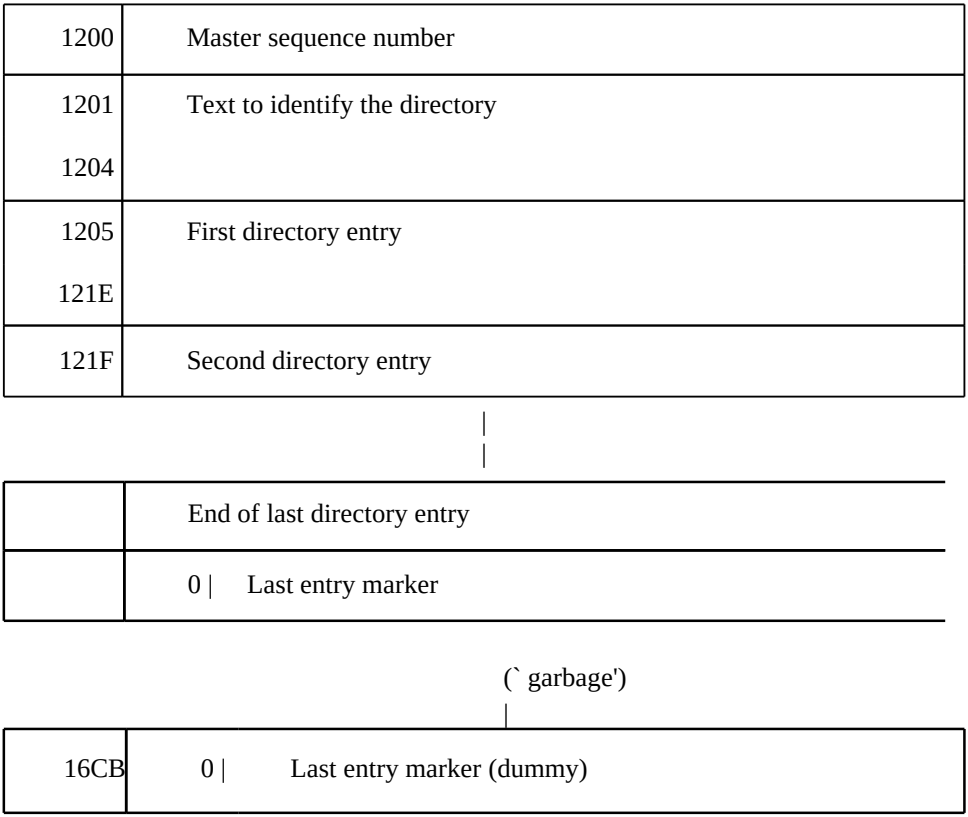
Directory information

A directory consists of five contiguous sectors on the disc drive. It contains a maximum of 47 entries, each entry consisting of 26 bytes as follows:

Name and access string	10 bytes
Load address	4 bytes
Execution address	4 bytes
Length in bytes	4 bytes
Start sector on drive	3 bytes
Sequence number	1 byte
Total	26 bytes

The remaining 58 bytes in the directory are one zero byte, one byte which is the directory master sequence number, 19 bytes of directory title, three bytes for the parent pointer (ie the disc address of A), a directory name string, and a directory identity string. The master sequence number is incremented every time the directory is rewritten. When an entry is made or changed in the directory the entry's sequence number is set to the directory master sequence number.

The currently selected directory is stored in RAM from &1200 to &16FF when ADFS is selected. The attributes are stored in the top bit of the first four characters of the entry name, so the **R** attribute of the first entry is in bit 7 of &1205, **W** in bit 7 of &1206, **L** in &1207 bit 7, **D** in &1208 bit 7. **R** of the second entry is in bit 7 of &121F and so on. The end of the list of entries is denoted by a 0 in the first character position of the first unused entry, hence the 0 before the directory name. A store map of locations &1200 to &16FF is shown below.



16CC 16D5	Directory name
16D6 16D8	Parent pointer (LSB) (MSB)
16D9 16F9	Directory title
16FA	Master sequence number
16FB 16FE	Text to identify the directory
16FF	Reserved

Location &1200 in the above example contains byte 0 of the first sector of the directory (sector 2 for directory $). Location &16CB contains byte &CB of the fifth sector of the directory (sector 6 for directory $).

12 Filing system command summary

Command	Minimum abbreviation	Purpose
***ADFS**	***A.**	Starts the ADFS from another filing system, without the user having to press **BREAK**
***ACCESS**	***A.**	Allocates object attributes.
***BACK**	***BAC.**	Goes back to previously selected directory.
***BYE**	***BY.**	Closes all sequential access files, and moves read/write heads to shipping zone.
***CAT**	***.**	Displays a catalogue of the CSD or a named directory.
***CDIR**	***CD.**	Creates a new directory.
***CLOSE**	***CL.**	Closes all sequential access files.
***COMPACT**	***CO**.	Compacts information on the disc.
***COPY**	***COP.**	Copies a list of files into another directory.
***DELETE**	***DE.**	Deletes a named object from the disc.
***DESTROY**	***DES.**	Deletes a number of objects from the disc in a single operation
***DIR**	***DIR.**	Selects a new currently selected directory.
***DISMOUNT**	***DISM.**	Ensures data on to the drive.
***EX**	***EX**	Displays information about files in a named directory or the currently selected directory.
***EXEX**	***E.**	Reads information in a specified file a byte at a time as if it were being typed in at the keyboard.
***FADFS**	***FA.**	Starts ADFS from another filing system without there having to be a disc in the drive.

***FREE**	***FR.**	Displays the amount of free space left on the disc.
***HELP**	***H.**	Displays useful information. *HELP ADFS displays the ADFS commands with syntax guidelines.
***INFO**	***I.**	Displays information about a list of objects.
***LCAT**	***LC.**	Displays the current library catalogue.
***LEX**	***LE.**	Displays information about the current library.
***LIB**	***LIB**	Sets the library to the specified directory.
***LOAD**	***L.**	Reads a file from disc to memory.
***MAP**	***MA.**	Displays a map of the free space on the disc.
***MOUNT**	***MOU.**	Initialises a drive.
***OPT1**	***O. 1**	Switches screen messages which accompany disc accesses on or off.
***OPT4**	***O. 4**	Sets the auto-start option of the currently selected drive.
***REMOVE**	***RE.**	Deletes a single named object from the disc, with no error reporting if the object does not exist
***RENAME**	***REN.**	Changes a specified object name, moving it to another directory if required.
***RUN**	***R.**	Runs a machine code program.
***SAVE**	***S.**	Saves a specified part of memory to the disc.
***SPOOL**	***SP.**	Transfers all text subsequently displayed on the screen into a specified file.
***TITLE**	***TI.**	Sets the title of the currently selected directory.

Appendix A

Fitting the ADFS ROM

You will need to fit the ADFS ROM to your microcomputer before you can use the Winchester disc drive. The instructions listed below should be read through before you carry them out; if you don't feel confident that you can fit the ROM, you should take it (together with your microcomputer) to your dealer, who will fit it for you.

The ADFS ROM should be fitted in a spare ROM socket The ROM sockets are located on the front right hand side of the circuit board inside the BBC Microcomputer casing (see diagram).

1. To get to the board, undo the four screws which hold the casing together— on some computers these will be marked 'FIX'. Two of these screws are at the back of the computer, and the other two are underneath near the front

2. Once the top is removed, undo the bolts holding down the keyboard assembly. These are located on either side of the keyboard. Some machines have two bolts, others may have three.

3. Refer to the diagram. Carefully lift the keyboard assembly until it is just clear of its locating lugs, rotate it in a clockwise direction until the five ROM sockets on the main circuit board are fully exposed, then rest it on the lower casing.

4. Examine the five ROM sockets. The one on the left contains the operating system. The BBC BASIC ROM is identified by the sequence B01' or B05' at the end of the second row of lettering printed on top of it, and should be found in the right hand socket If it isn't then it should be removed from its present socket (see below) and replaced in the correct position The ADFS ROM can be inserted in any one of the middle three sockets, depending on the priority you wish to assign to it

If you wish ADFS to be entered when you switch your microcomputer on (or on a hard reset) then the ADFS ROM should be inserted in the rightmost of the middle three sockets (if you don't, then the ROM may be inserted in any of the middle three sockets). If the socket where you wish to insert ADFS already contains a ROM it should be removed and inserted in another free socket

Removing a ROM

To avoid bending any of its pins, the ROM should be removed very carefully. Take a screwdriver or similar tool and gently prise up each end, a bit at a time.

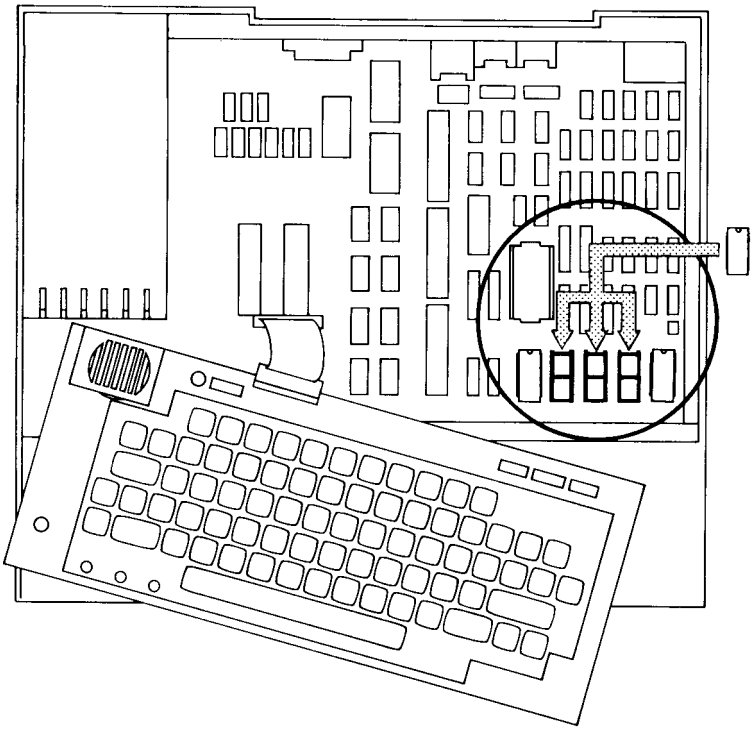
Fitting a ROM

1. Locate the half-moon cut-out at one end of the ROM. This end should face away from the keyboard when the ROM is inserted.

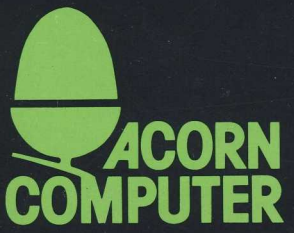
2. Holding the ends of the ROM between finger and thumb, line up all the pins over the destination socket Now press the ROM gently but firmly into its socket *Don't force it!* When it is all the way in it appears to be slightly raised. Check that all the pins have entered the socket, and that none are bent out or underneath.

Inserting the ADFS ROM

The diagram shows a plan view of the BBC Microcomputer with the casing removed. The ADFS ROM can be inserted in any of the highlighted ROM sockets.



Index



Acorn Computers Limited, Fulbourn Road, Cherry Hinton, Cambridge CB1 4JN, England