ACORN COMPUTER

Advanced Disc Filing System USER GUIDE

Advanced Disc Filing System User Guide

Addendum 1

Please note the following change to this guide:

Chapter 9 - The Filing System Utilities

The ADFS Utilities Disc supplied is an enhanced version of the disc described in chapter 9. The number of utilities has been increased and the disc incorporates a menu for ease of selection.

The utilities are each described in detail on the disc by means of a '**Help**' facility and are summarised below:

***AFORM**
Formats a floppy disc in ADFS format.

***BACKUP**
Copies all the contents of one disc onto another.

CATALL
A BASIC program which produces a listing of the contents of all the directories on a disc in a similar form to that given by the ***CAT** command.

COPYFILES
A BASIC program used to copy files from one filing system to another.

DIRCOPY
A BASIC program which copies the entire contents of a specified ADFS source directory, and all its sub-directories, to a specified destination directory.

EXALL
Similar to CATALL, but displays information similar to that given by the ***EX** and ***INFO** commands. A BASIC program.

HARDERROR
This BASIC program enables permanent floppy disc errors to be ignored by the ADFS, thus avoiding the "Disc Error" message which would be generated when the bad track is accessed.

RECOVER
This BASIC program enables a file, or part of a file, which has been deleted accidentally using the ***DELETE** command, to be recovered.

***VERIFY**
Verifies a disc by checking each sector to determine if it is readable.

To use the Utilities disc, insert it into the drive (the top drive on dual drives) and, whilst holding down **SHIFT** and **A**, press and release **BREAK**. The Utilities Menu will appear, as follows:

```
                    Utilities Menu

f0 - Aform
f1 - Backup
f2 - Catall
f3 - Copyfiles
f4 - Dircopy
f5 - Exall
f6 - Harderror
f7 - Recover
f8 - Verify
f9 - End

Press :-
    Function keys to run Utilities
Shift-Function keys for help
```

where **End** will stop the menu program and return you to BASIC.

When using the Help facility (**SHIFT**+**<function key>**), use **SHIFT** to scroll to the end of the text.

To use individual programs, without recourse to the menu, insert the disc into a drive and select the drive using a ***DIR** or ***MOUNT** instruction. Type ***LIB. LIBRARY** to set the library. For machine-code programs, simply type:

***<filename> RETURN**

as indicated above. For BASIC programs, type:

CH. "$ UTILITIES.<filename>" RETURN

The Advanced Disc Filing System User Guide

Part no 0433,070
Issue no 1
Date August 1985

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

First published 1984
Published by Acorn Computers Limited
Typeset by Interaction Systems Limited, Cambridge

Contents

Introduction

Equipment required

The ADFS upgrade consists of this User Guide, the ADFS ROM (or EPROM), a utilities disc and a guarantee card.

If any of the necessary items are missing then contact your dealer.

To use ADFS you will need:

– A single or dual disc drive and a connecting cable, terminated in a 34 way IDC socket. A second cable will be needed to supply power to the disc drive, either from the A.C. mains supply if it is a self-powered drive, or from the Auxiliary power connector of the computer.

– A BBC Model B or Model B+ Microcomputer fitted with a 1770 disc interface and the Advanced Disc Filing System ROM.

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. This guide describes the operation of ADFS with floppy disc drives. ADFS will also control an ACORN fixed 'hard' disc (Winchester) drive. For details, see the *ACORN Winchester Disc Filing System User Guide*.

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 are the same size as the information they contain. A file has a name (decided by you) and an address (decided by the computer), which means that it's 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'd determined which grid square it was in. The computer needs a 'grid' on 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 Fig 5), and must be carried out before information can be stored on a new, blank disc. Once a disc has been formatted it stays formatted; you don't have to re-format 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'.

What a disc drive does

When you insert a disc into the disc drive and close the drive door, the disc spins inside its protective jacket (don't confuse the protective jacket with the disc envelope – see Fig 2). When you want to access 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 (which actually rests on the surface of the disc) along the 'head slot' in the disc jacket to the sector on the disc where the start of the information in the named file is recorded. While this is going on, the red light at the front of the drive will come on. You must not remove a disc while the red light is on.

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 ADFS floppy disc system, you must fit the ADFS ROM to your microcomputer and then connect the disc 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

With the power turned off, connect the two cables from the disc drive to the underside of the computer as shown in Fig 1. Both connections have been designed to make it extremely difficult to fit them the wrong way up. The plug on the narrow (power) cable is shaped like a rectangle with two adjacent corners cut off, the socket into which it fits being similarly shaped. Do not force it in the wrong way round. The socket on 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 socket is with the arrowhead at one end of the socket aligned with the arrowhead next to the 'disc drive' label on the computer. When the drive is connected, turn on the power.

Starting the filing system

If you've not already switched the computer on then do so now. The following message (or one very similar) should appear on the screen:

`BBC Computer 32K` or `ACORN OS 64K` or your normal banner message

```
Acorn ADFS
```

`BASIC` or your default language

`>_` or your default language cursor

If 'ADFS' does not appear anywhere in the message (indicating that a ROM other than the ADFS ROM is in the rightmost ROM socket), hold down **CTRL** and `A` together and then press and release **BREAK**. The message should then change to that shown above, indicating that the ADFS is now ready for use.

At the same time the disc drive will start making a whirring noise and the red light at the front will come on (the top light if you have a dual drive; if both lights come on then it probably means that the broad ribbon cable has been connected the wrong way round). This shows that the drive is properly connected and working and that it is trying to read a disc (if nothing happens you should check the connections to the computer). To stop the drive press the **ESCAPE** key.

For a system containing a floppy drive and a Winchester drive

For use with the ACORN Winchester drive, see the Winchester Disc Filing System User Guide.

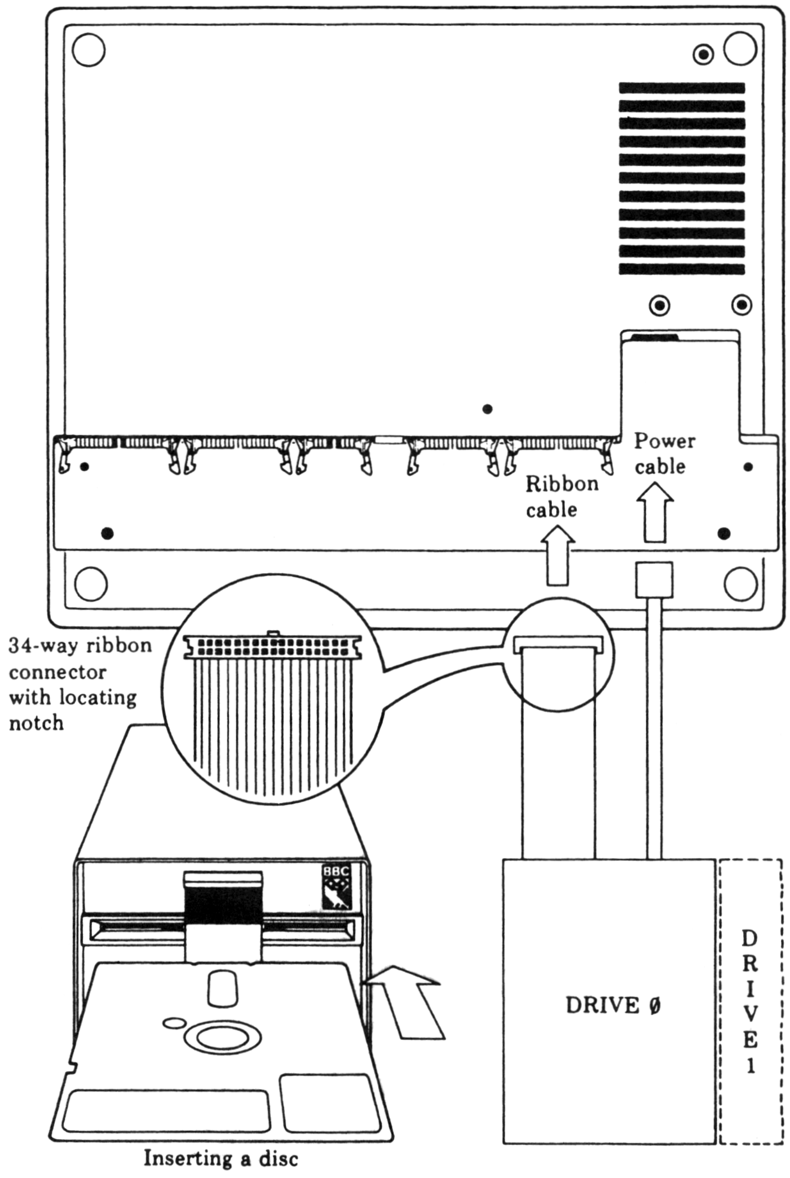
If you want to select the floppy drive type

```
*DIR :4
```

and press **RETURN**. The disc drive (the top drive if you have a dual drive) will start making a whirring noise and the red light at the front will come on (if both lights come on then it probably means that the broad ribbon cable has been connected the wrong way round). This shows that the drive is properly connected and working and that it is trying to read a disc (if nothing happens you should switch off and check the connections to the computer). To stop the drive, press the **ESCAPE** key.

Now turn to chapter 3, which explains more about discs and how to use them.

Figure 1 Connecting the disc drive



3 Discs

Care of the disc unit

The disc unit is a fairly robust device and should give good service in normal use. The only guidelines to be observed are that the unit should not be exposed to excessive heat, moisture, direct sunlight, or very dusty conditions. Although most people would use the disc unit with it standing on its feet, it can be safely operated in any convenient position.

Disc capacity

The disc unit supplied for the BBC Microcomputer uses 5¼″ discs for storing information. A single disc (80 track double-sided) can store up to about 640K bytes of data, equivalent to about 300 pages of text (each page the same size as in this book). Some of the storage space is reserved for use by the computer itself; this is detailed in chapter 10.

Handling

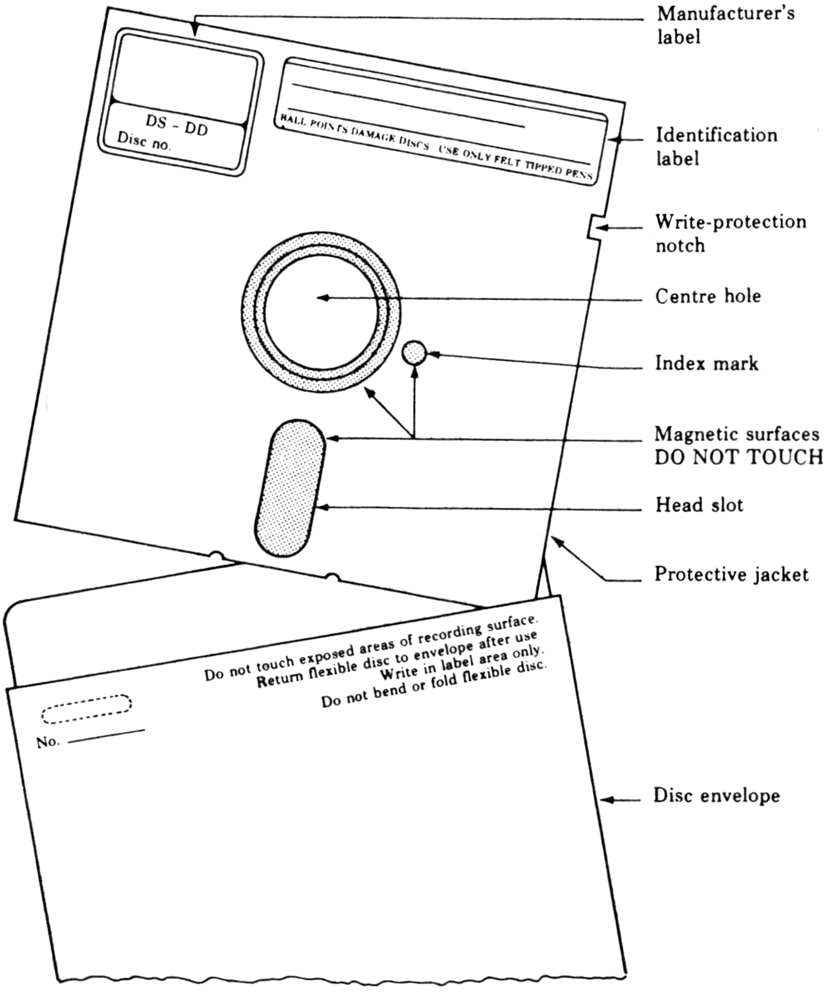
Discs should be handled with care to avoid physical damage or damage to the recorded information. Fig 2 shows the various parts of the disc that we'll be referring to. The following guidelines should be observed:

- Don't try to remove the circular magnetic disc from the square black protective jacket covering it.
- Don't touch the exposed recording surfaces.
- Avoid getting the discs dusty by putting them in their envelopes when they are not in the disc drive.
- Do not bend the discs, drop them on the floor, or put heavy objects on them.
- Keep the discs in a storage box designed for the purpose.
- Keep the discs away from strong magnetic fields such as those generated by televisions, monitors, tape recorders, telephones, transformers, calculators etc. You should also not put discs on top of the disc drive or the computer.
- If you write on the disc labels, do so before you stick the label to the disc sleeve.
- Insert the disc into the drive carefully. If it rotates noisily open the drive door and adjust the disc. If you get an error message such as Disc fault or Drive fault, then the disc probably hasn't centred in the drive. To overcome this, insert the disc and access the drive (ie type in any command which would cause

the drive to operate, eg *MOUNT) before closing the drive door. This is especially important when a new disc is inserted.

These guidelines are deliberately comprehensive, but don't let them put you off. Handled sensibly, a disc will give good service.

Figure 2 A 5¼″ disc

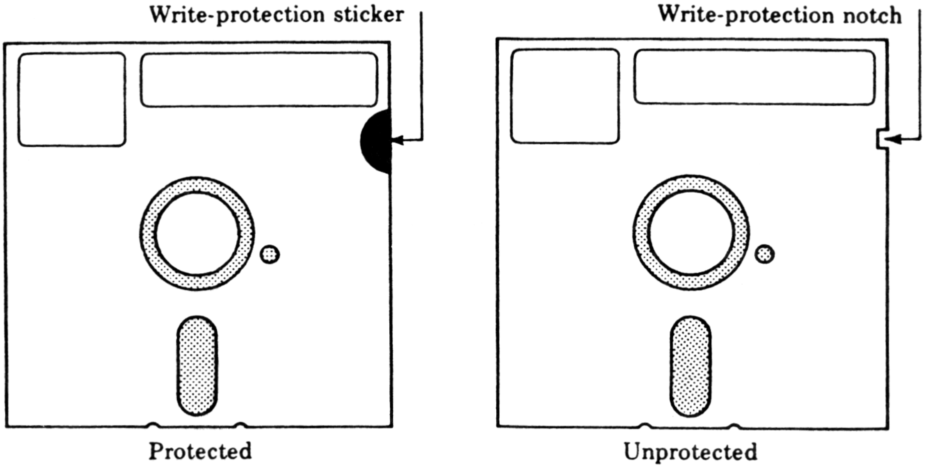

DS - DD
Disc no.
BALL POINTS DAMAGE DISCS USE ONLY FELT TIPPED PENS
Manufacturer's label
Identification label
Write-protection notch
Centre hole
Index mark
Magnetic surfaces DO NOT TOUCH
Head slot
Protective jacket
Do not touch exposed areas of recording surface.
Return flexible disc to envelope after use.
Write in label area only.
Do not bend or fold flexible disc.
No.
Disc envelope

Preventing accidental erasure

You'll notice from Fig 2 that there is a small notch in the side of the protective jacket called the 'write protection' notch. Every box of discs is supplied with a number of adhesive tabs which can be used to cover the notch. Sticking a tab over the notch such that it completely covers it will prevent the disc drive from writing to the disc or deleting anything on it; any attempt at writing or deleting will result in the

```
Disc protected
```

error message. Reading the existing information on the disc will still be possible.



Copying information

Another way of preventing the loss of important information is to keep several copies of it on different discs. Where computers are used in business, industry, or other activities which use large volumes of information, a standard 'archiving' routine has been evolved. It is often called the 'grandfather, father, son' system of copying information, and is good practice for anyone using a disc storage system. It works as follows:

Day 1, MASTER copied to GRANDFATHER
Day 2, MASTER copied to FATHER
Day 3, MASTER copied to SON

As you can see, it involves keeping three separate discs, each with a copy of the information from the master disc on it. On day 4 the master would be copied to the grandfather again and so the cycle continues. In business, where information on a disc might change from day to day, this regular routine is important. (It takes time, but not nearly as much time as it would take to re-input a lot of lost information!) For personal computing such a system may not be essential, but it still makes sense to keep copies of programs and data which you have worked hard to produce. The filing system's `*COPY` command can be used to copy information from one disc to another (including to a Winchester disc, if you have one). One disc you have which is well worth copying is the utilities disc. The following procedure tells you how to copy it.

First of all, if you have a Winchester disc unit, disconnect it; this will make things simpler. Before the copy can be made, the disc onto which you're going to copy the utilities disc must be formatted, as we mentioned in chapter 1. Formatting is explained in more detail later on; all you need to know at this stage is that a disc must be formatted before information can be put onto it. Insert the utilities disc into the drive (or into the top drive if you have a dual drive unit) and close the drive door. Now press **BREAK** while holding **CTRL** down. When the message:

`BBC Microcomputer 32K` or `ACORN OS 64K` or your normal banner message

```
Acorn ADFS

BASIC                      or your default language
>_
```

(or something very similar) is displayed, type

`*AFORM` **RETURN**

In answer to the question that is displayed type

`0` **RETURN**

and in answer to the next question simply press **RETURN**. A list of options is displayed; type in the appropriate letter (or number) according to the type of disc drive you have and press **RETURN**. Before answering the next question, take out the utilities disc and insert a blank disc. The question currently being displayed just asks you to confirm your intention to format the disc (this gives you a second chance to make absolutely sure that you've got the right disc in). Assuming you have, type

`YES` **RETURN**

(any other answer will abort the formatting sequence). A constantly increasing 2-digit number appears on the screen accompanied by the message

```
Formatting..
```

followed by

```
Verifying...
```

and finally by

```
Verification complete

>_
```

('Verification' is a process which checks that the disc has been formatted correctly.) You are now in a position to make a copy of the utilities disc. The procedure to be followed depends on whether you have a dual disc drive or a single disc drive. If you have a dual drive, insert the utilities disc into drive 1 (ie the bottom drive), close the door and type

`*COPY :1.* :0` **RETURN** or `*BACKUP 1 0` **RETURN**

The disc drives will show signs of activity and after a few seconds the 'prompt' line, ie

```
>_
```

will appear, indicating that the copying operation has been completed. The difference between the above two commands is that `*BACKUP` copies not only the files on the utility disc, but all its free space as well; this means that any files which are already on the backup disc will be overwritten (see Fig 3). Backing up is a very important procedure, but the latter point must be remembered. The second * character in the `*COPY` instruction means 'all files', so the instruction is simply saying 'copy all files on the disc in drive 1 onto the disc in drive 0'. (What actually happens in both cases is that the data is transferred from the first disc to the computer's memory, and then from the computer's memory to the second disc.) You can also copy single files or groups of files from one disc to another. See **Fig 3** and the appropriate section of chapter 5 for more details.

The `*COPY` command cannot be used to make copies from one disc to another if you only have a single disc drive; in this case the utilities disc has to be copied using `BACKUP`, which is of one the programs held on the utilities disc itself. Take the blank disc out of the drive, insert the utilities disc, close the drive door and type

`*BACKUP 0 0` **RETURN**

A message will be displayed asking you to insert the source disc into drive 0. The source disc is the utilities disc, which is already in drive 0, so in fact you need take no action. Having pressed **RETURN**, wait for the next message. In answer to this, you should remove the utilities disc from the drive and insert the blank disc; pressing **RETURN** will then cause information to be written onto the disc. This process will have to be repeated several times before all the information is copied. During the backing up process it is a good idea to put a write-protection tab on the source disc, so that if you do make a mistake when swapping the discs you won't destroy the source disc.

The numbers in the above instructions are drive numbers; these are as follows:

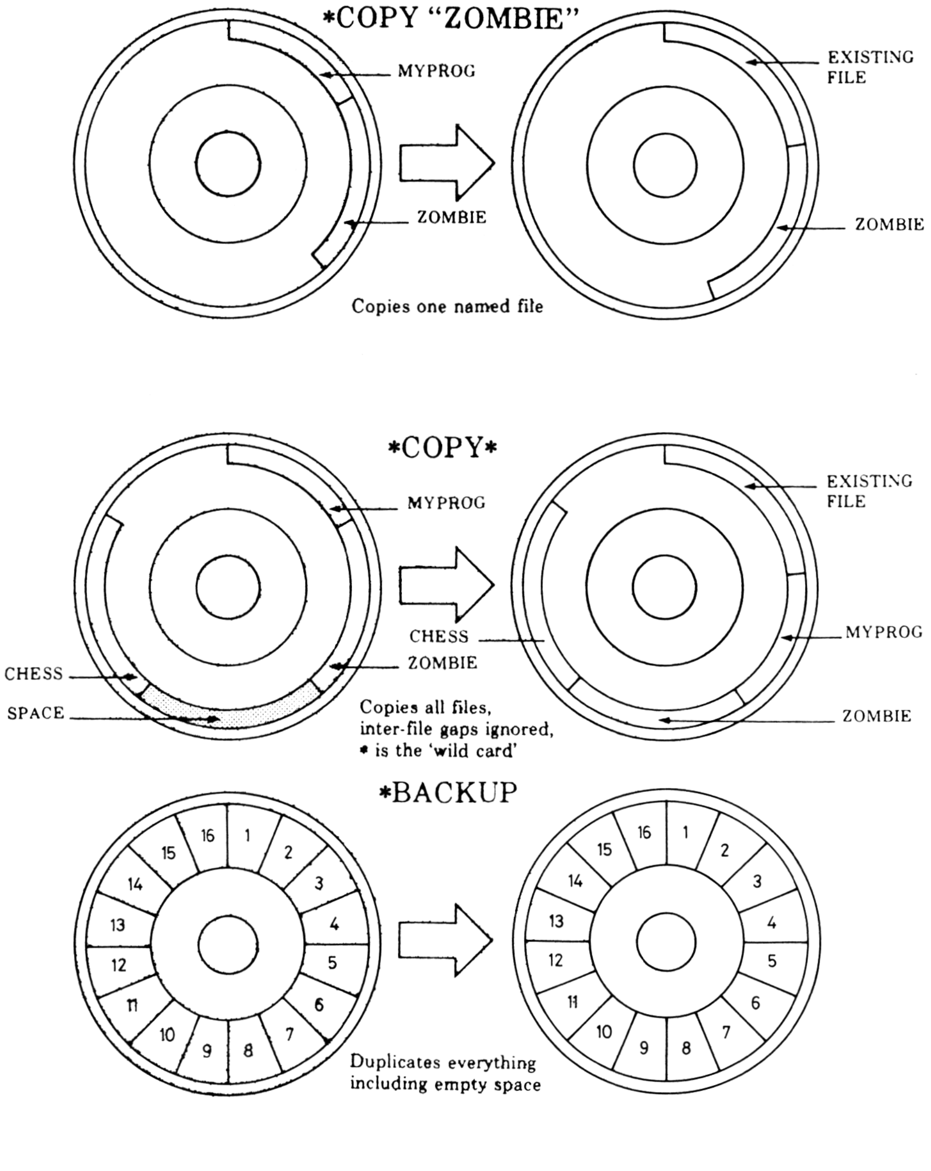
Floppy drives only

Dual drive:	upper drive 0	Single drive:	drive 0
	lower drive 1		

Floppy drives and a Winchester drive

Dual drive:	upper drive 4	Single drive:	drive 4
	lower drive 5		
Winchester drive:	drive 0		

Figure 3 Copying



Tracks, sectors, and bytes (see Fig 4)

Information is written on to the disc in concentric circles called tracks. Each track is divided into 16 sectors, each of which can contain 256 bytes of information. The total storage capacity of an 80 track double-sided disc is therefore made up of:

160 tracks × 16 sectors × 256 bytes = 655,360 bytes

Formatting

As mentioned previously, before a new disc can be used it must be formatted. The formatting process includes setting up the track and sector format on the disc and creating a 'master' directory from which all the data on the disc can be accessed (see 'root directory' in chapter 4). Discs are formatted using the AFORM utility program, see chapter 9.

Figure 4 Tracks, sectors, and bytes

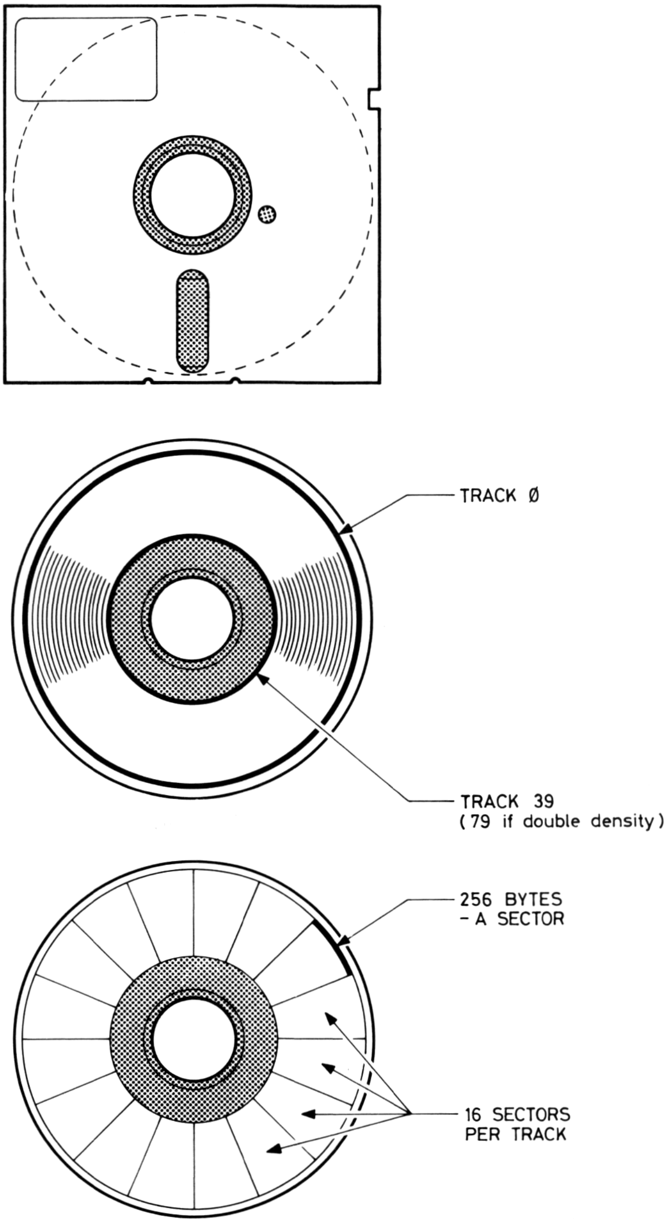
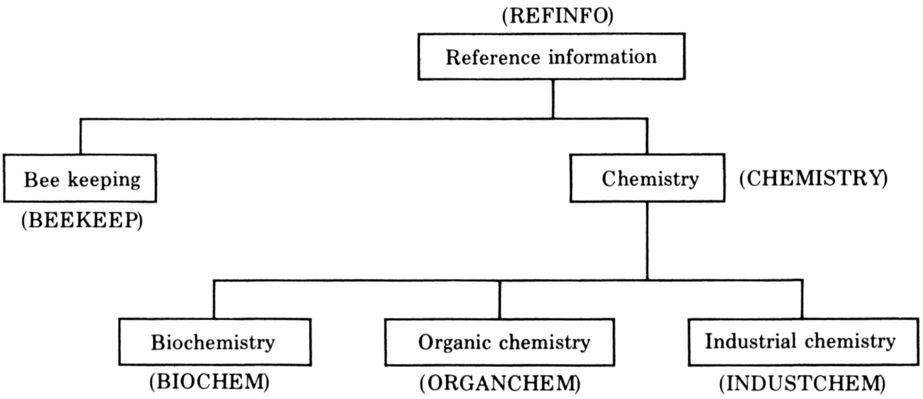


Figure 5 A typical hierarchical filing system



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 5). 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.

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 5. 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) Memo1, it may not be enough to type, for example,

`LOAD "Memo1"` **RETURN**

since Memo1 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 Memo1 at the bottom right of figure 6 is

LETTERS.BIZLETS.Memo1

so to load Memo1 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

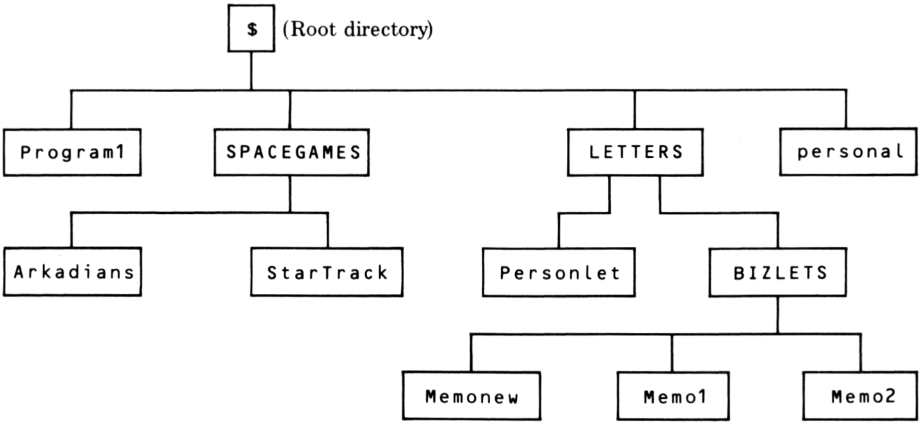
A directory is a collection of objects (up to 47). It can be part of another directory and can itself contain other directories. A directory name can be up to ten characters long; any characters can be used except

* . : $ & @ ^ or a space

(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 6 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 `$` symbol (or the `&` symbol) is used to refer to the root directory (you can't change this). It is created (empty) when the disc is first formatted, and is accessed whenever the disc filing system is first entered. At this point the root directory is said to be the 'currently selected' directory (the CSD). Refer to figure 6; if you asked the computer for a list of all the objects in the CSD (ie `$`), 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 6, 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 6, 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 `Personlet`.

@ 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

When a *command of the form `*<filename>`, which is not an ADFS filing system command, is passed to the ADFS, it will first search the Currently Selected Directory (CSD) for a file of that name. If it cannot find that file within the CSD, it then searches another directory – the currently selected library directory.

The ADFS enables you to specify one directory as the library. The default library is set up as follows:

– If ADFS is entered with a **CTRL** A **BREAK** and there is a directory in `$` whose name begins with `LIB`, that directory is set as the library.

– If ADFS is entered as above, but there is not a directory whose name begins `$.LIB`, `$` (ie the root directory) is set as the library.

Note, however, that the library directory may be 'unset' following a `*DIR` or `*MOUNT` 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 reset by another `*LIB` command).

The library directory enables you to store any frequently used machine-code programs, such as utilities, in one central location. They may be accessed as required, whether or not the library is the CSD and whether or not (for systems containing more than one drive) the library is on the currently selected drive.

For example:

The currently selected library directory `:0.$.LIB1` (on drive 0) contains a utility program `MC1`.

Your CSD is `BIZLETS`, on drive 1 and you wish to use the utility program MC1.

When you type

```
*MC1
```

the command is passed to the ADFS which first searches your CSD (`BIZLETS`) for `MC1`, which is not present. It then goes to the library directory `$.LIB1` on drive 0. `MC1` is found here, and the program is run.

`BIZLETS` remains as your CSD.

NOTE: A library filename should be unique. If it is not, you may inadvertently be in a CSD containing a file or directory with the same name as, say, a utility file in the library directory. If you use a *command to access the utility file in the library, the system will find the file or directory which is in the CSD first, not the utility file you require.

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 CSD,

`*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*.*O#` **RETURN**

where `*O#` means 'all names with `O` 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 (&1D00 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.

Changing discs

One important fact to realise (especially if up to now you've been using discs under another filing system), is that the ADFS stores disc directory information in the computer RAM (see chapter 11 for full details), and that this information will remain there, even after the disc has been removed. Replacing one disc with another and then typing (for example)

```
*CAT
```

will result in directory information for the first disc (ie the one you've just taken out) being displayed. This is because the computer has not been told that the first disc has been removed. The filing system includes commands for use in this situation. The computer should always be told about a disc change before the disc is removed (see `*DISMOUNT` in chapter 5), unless you have only being looking at what is on a disc (eg by using `*CAT $`), in which case `*DISMOUNT` is unnecessary.

Drive numbers, the Winchester disc system, and future expansions

If your disc storage system does not include a Winchester disc drive, then the floppy drive numbers under ADFS are as follows:

Drive number	Disc unit
0 or 4 or A or E	floppy single drive, or top drive of floppy dual drive
1 or 5 or B or F	bottom drive of floppy dual drive

Note that for your convenience drive 'letters' are provided as an alternative to drive numbers. Note also that ADFS does not assign a separate drive number to each side of a disc.

If you're operating your floppy drive in conjunction with a Winchester disc drive, drive numbers are 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	floppy drive 0
5 or F	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 floppy drives.

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 dual floppy drive and drive 0 (ie the top drive) is the currently selected drive, then typing in

`LOAD ":1.MYPROG"` **RETURN**

would load file MYPROG from the root directory of drive 1 (ie the bottom drive). Similarly, all filing system object specifications, wildcard object specifications and list specifications can begin with a drive number. For example,

`*COPY :1.MYPROG :Ø` **RETURN**

would copy MYPROG in the root directory of drive 1 to the root directory of drive 0. (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. If you only have a single drive then a drive number need not be included; although

`LOAD ":Ø.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
<drv>	= drive number

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

For <obspec>, see 'Wildcard facilities', chapter 4.
For <listspec>, see 'Multi-object operations', chapter 4.

*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 `*LOAD`ed, 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 a program called `PROG` from (say) cassette tape, typing

```
*ADFS             (if ADFS is not the current filing system)
*MOUNT <drive>    (if the drive required has not previously been specified)
SAVE "PROG"
```

would save the program onto disc. (Note that the program could not necessarily be made to run; see 'DFS-ADFS' 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 `*BACK`s 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 a 'new' 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. This variation of the `*COMPACT` command will not work on a Model B+ Microcomputer with shadow screen selected; in shadow mode, the apparent screen memory is 0 bytes and an error message will result. It is necessary, therefore, to always specify `*COMPACT <SP> <LP>` when using shadow screen.

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
0004B6  :  000002
0006B9  :  000003
00078C  :  000002
000A97  :  000001
000EFA  :  008E4E
```

Typing

```
*COMPACT
```

followed by

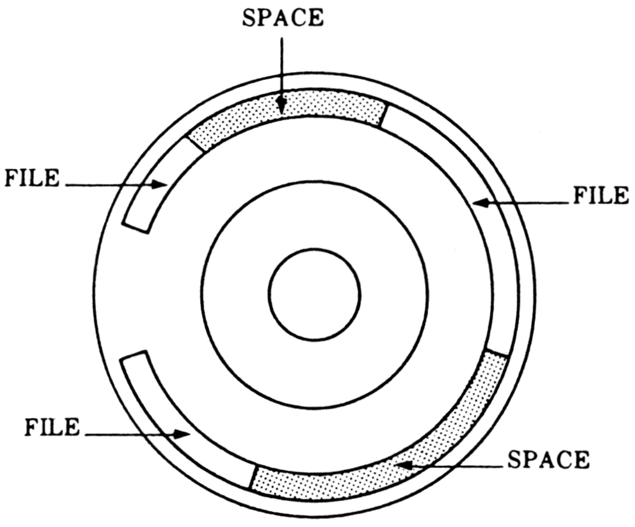
```
*MAP
```

would give

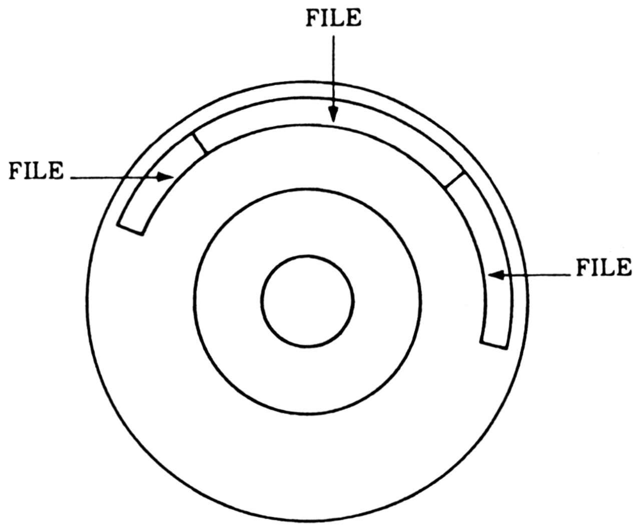
```
Address :  Length
0006B9  :  000003
0007CF  :  00000A
000EFA  :  008E4E
```

(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.)

Fig 7 `*COMPACT`



Before



After

*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* $.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 `*DELETE`d 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 CSD).

```
*DIR ^
```

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

Notes

The `*DIR` command should also be used following a disc change. Assuming the new disc has been inserted into the drive previously occupied by the old one, typing

```
*DIR
```

will cause the root directory and free space map of the new disc to be loaded into RAM, thereby enabling all data on the disc to be accessed. If however the old disc had some open sequential files on it which had not been closed then the above procedure would produce an appropriate error message, and data could have been lost. See `*DISMOUNT`.

Note that typing

```
*DIR :<drv>
```

produces the same effect as typing

```
*MOUNT <drv>
```

If the library is currently set on the drive into which a new disc is being inserted, it will be 'unset' by `*DIR` (or `*MOUNT`). The library may be set using `*LIB` - see chapter 4. The library, whether it is set or unset, is unaffected if `*DIR` (or `*MOUNT`) is used on any drive other than the drive on which it is set.

*DISMOUNT (<drv>)

Purpose

To 'ensure' data onto 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 must be used prior to changing a disc on which you know there is, or on which you suspect there may be, some open sequential files. (Obviously, any program which opens sequential files will (or should) contain instructions to close them again. However, should such a program exit for any reason prior to finishing its run, the 'closing' instructions may not be accessed.) Typing

```
*DISMOUNT
```

is equivalent to typing the sequence:

```
*CLOSE
*DIR $
```

and, if the currently selected library is set on that drive,

```
*LIB $
```

Examples

```
*DISMOUNT
```

Ensure data onto the currently selected drive.

```
*DISMOUNT Ø
```

Ensure data onto drive 0.

Description

Ensures data onto 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 `*DISMOUNT`ed drive, the system is put into a state such that the CSD is 'unset'. The library directory will be 'unset' if it is on a `*DISMOUNT`ed drive, but unaffected ('set' or 'unset') if it is currently on another drive. The next access to the `*DISMOUNT`ed drive will either produce the

```
No directory
```

error message, or directory `$` will be read from drive 0 (`:0.$`) and treated as the CSD, according to the nature of the access. If drive 1 is the currently selected drive, typing

```
*DISMOUNT 1
*CAT
```

would produce

```
                          (19)
Drive:0                   Option 00 (Off)
Dir. "Unset"              Lib. "Unset"   - but see above

LIBPROG    DLR (22)   PROGX       WR (08)   PROGY       WR (19)
```

The CSD is not set, but is treated as being `:0.$` since this is the default directory in the `*CAT` command. However, typing

```
*DISMOUNT 1
LOAD "PROGX"
```

would produce the

```
No directory
```

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

Should a disc be removed which should have been `*DISMOUNT`ed first, the error message

```
Disc changed
```

may be displayed when the new disc is accessed. If this happens, the system should be restarted with a hard break, as data will have been corrupted.

*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     00007B
FT             DLR(01)          00000C
ROB            DLR(00)          000007
X              WR (19)          00000000     00000000     0000C580     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   (19)
```

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

```
*FADFS
*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. $                  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 CSD). 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
0006B9  :  000003
0007CF  :  00000A
000EFA  :  008E4E
```

Description

Displays the free space map.

Associated commands

```
*COMPACT
*FREE
```

*MOUNT (<drv>)

Purpose

This command initialises a drive (ie reads the free space map and stores the appropriate addresses in RAM).

Example

```
*MOUNT 0
```

Initialise floppy drive 0 (or the Winchester drive if one is present).

Description

Initialises a drive.

Associated commands

```
*DISMOUNT
```

Notes

`*MOUNT` can be used to swap between drives (if you have more than one) or can be used, following a disc change, to enable the new disc to be accessed. Note that typing

```
*DIR :<drv>
```

produces the same effect as typing

```
*MOUNT <drv>
```

If the library is currently set on the drive into which a new disc is being inserted, it will be 'unset' by `*MOUNT` (or `*DIR`). The library may be set using `*LIB` – see chapter 4. The library, whether it is set or unset, is unaffected if `*MOUNT` (or `*DIR`) is used on any drive other than the drive on which it is set.

*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 Ø` or `*OPT 1,Ø`

disables the messages.

Description

Message system to display file information at every access.

Associated commands

```
*INFO
```

Notes

A space or a comma between `*OPT 1` and its argument (n) is essential.

```
*OPT Ø
```

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

*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 `$`) 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.

Examples

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
OPENIN
OPENUP
PTR#
EXT#
INPUT#
PRINT#
BGET#
CLOSE#
BPUT#
EOF#
```

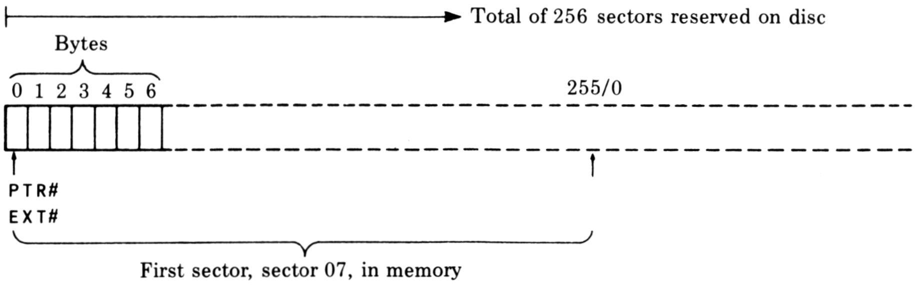
To prepare a file to receive data the `OPENOUT` keyword is used. In the *User Guide* the following example is given:

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

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

1. A file called `cinemas` is entered into the currently selected directory.
2. 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 not created and an error message is displayed.
3. Evaluating `PTR#` and `EXT#` at this point will reveal that they are both set to zero.
4. 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'.

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

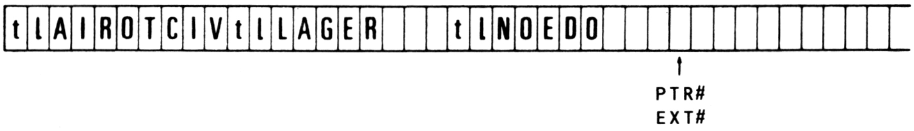

Total of 256 sectors reserved on disc
Bytes
0 1 2 3 4 5 6
255/0
PTR#
EXT#
First sector, sector 07, in memory

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 'l', 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 'l' to give the length. Real numbers are stored in exponential format, integers are stored with the high order bytes first in the file.

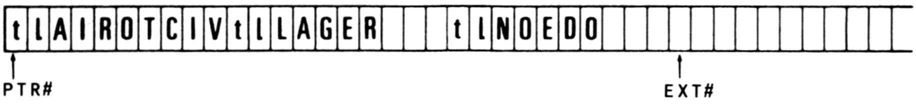
In the example we have used only the first 26 bytes of the file, so everything written to the file fits into the first sector which is in a 'buffer' in memory. If we had gone on writing names, the filing system would eventually have put the information in the memory buffer on to sector 07 of the disc 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).

NOTE: If the currently selected directory already contains a file called `cinemas`, it will be deleted.

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	&021C	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	&021A	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 &021C).

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 OR'ed with the current drive number to give the drive number to use. The function code field is structured as shown below:

Value	**Meaning**
&08	Read
&0A	Write

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 &000567 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	&05	
08	&67	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 appropriate type of error. The channel number returned is in hexadecimal.

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`	The ADFS filing system
`*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
`*TELETEXT`	The TELETEXT filing system
`*TELESOFT`	The TELESOFT filing system
`*ROM`	The ROM filing system
`*PRESTEL`	The PRESTEL filing system

9 The filing system utilities

The floppy disc system includes a number of 'utility' programs (or just 'utilities' for short), supplied on floppy disc. The utilities are described in detail in the rest of this chapter, and are summarised below.

AFORM

Formats a floppy disc in ADFS format.

VERIFY

Verifies a disc.

BACKUP

Copies all the contents of one disc onto another.

ADFS-DFS

Copies files from an ADFS formatted disc onto a DFS formatted disc.

DFS-ADFS

Copies files from a DFS formatted disc onto an ADFS formatted disc.

DIRCOPY

Copies the entire contents of a specified source directory, and all its sub-directories, to a specified destination directory.

AFORM

Purpose

Formats a disc in ADFS format. The disc can be formatted as 40 track single-sided, 80 track single-sided, or 80 track double-sided.

How to use it

Insert the utilities disc into a drive, close the drive door and select the drive using a `*DIR` or `*MOUNT` instruction. AFORM is a machine code program, so to start it type

```
*AFORM <optional parameters>
```

followed by **RETURN**. The optional parameters are the drive number to be formatted and an indication of the disc capacity (ie 40 track single-sided, 80 track single-sided, or 80 track double-sided). The flexibility built in to AFORM allows either both parameters, one parameter, or no parameters to be specified. Supplying no parameters, ie just typing

`AFORM` **RETURN**

results in a question being displayed asking you which drive is to contain the disc to be formatted. If you are unsure how to specify the drive number, simply type **RETURN**; this results in a display of the drive number options. Remember that a single drive is always drive 0 (or 4 or A or E), and that the top and bottom drives of a dual drive are numbered 0 and 1 respectively (with the other numbers and letters as options). Having specified the drive number, you are asked to specify the 'size' of the disc you wish to format. Again, typing **RETURN** will display a list of options. For example, to format a disc as 80 track double-sided, type

```
160
```

or

```
L
```

(followed by **RETURN** in both cases). Note that a disc should only be formatted to match the type of disc drive you have; it would be of no use formatting a disc as 80 track double-sided if you only have a disc drive capable of handling 40 track single-sided discs, although you could (if you wished) do it the other way around.

Having specified the disc size, a question is displayed asking you to confirm that you wish to format the disc in the specified drive. This gives you a chance to change your mind, and also enables you to make sure that the disc you're about to format is the correct one. If the drive you have specified for formatting is the one currently occupied by the utilities disc you should remove the utilities disc at this point. If you're sure that you wish to go ahead with the formatting sequence, type in

`YES` **RETURN**

(any other answer will abort the formatting sequence). The disc drive will show signs of activity and the screen will display a message telling you what is going on (the constantly increasing 2-digit number corresponds to the track number (in hexadecimal) that is being formatted or verified). The sequence ends with the message

```
Verification complete

>_
```

indicating that the disc is ready for use.

As you become familiar with the formatting process you may wish to take advantage of some short cuts offered by AFORM. For example, if you wish to format a disc in drive 1, typing

`*AFORM 1` **RETURN**

would result in only the

```
Size of disc?
```

question being displayed (plus the final confirmation request). Alternatively, typing

`*AFORM L` **RETURN**

would only give

```
Format which drive?
```

Quickest of all, formatting an 80 track double-sided disc could be achieved by typing

`*AFORM 1 L` **RETURN**

or by

`*AFORM L 1` **RETURN**

(the order of the two parameters doesn't matter, but there must be a space in between them).

VERIFY

Purpose

Checks each sector of a disc for legibility. VERIFY can be used to check whether a disc is ADFS-formatted and error free. Verification is carried out automatically by AFORM.

How to use it

Insert the utilities disc into a drive, close the drive door, and select the drive using a `*DIR` or `*MOUNT` instruction. VERIFY is a machine code program, so to start it type

```
*VERIFY <optional parameter>
```

followed by **RETURN**. The optional parameter is the drive number to be verified. If you don't supply it, the drive number options are displayed together with a question asking you which drive you wish to verify. Having supplied the drive number and pressed **RETURN**, a message is displayed telling you what to do next. If you wish, you can specify the drive to be verified along with the `*VERIFY` command. For example, to verify drive 1 simply type

`*VERIFY 1` **RETURN**

BACKUP

Purpose

To read all the information from one disc and write it to another, producing two discs holding identical information.

How to use it

Insert the utilities disc into a drive, close the drive door, and select the drive using a `*DIR` or `*MOUNT` instruction. BACKUP is a machine code program, so to start it type

```
*BACKUP <optional parameters>
```

followed by **RETURN**. The optional parameters are the number of the drive to contain the disc to be backed up (the 'source drive', containing the 'source disc') and the drive to contain the disc to receive the information (the 'destination drive', containing the 'destination disc', which must be ADFS formatted).

If no drive numbers are supplied a message is displayed listing the drive number options available, together with a prompt asking you to type in the source drive number. Remember that a single drive is always drive 0 (or one of the related options), and that the top and bottom drives of a dual drive are numbered 0 and 1 respectively (with the other numbers and letters as options). Having typed in the source drive number, further prompts are displayed which are self explanatory.

The backup process can be streamlined by typing in the source and destination drives straight away. For example, to back up drive 0 to drive 1, simply type

`*BACKUP 0 1` **RETURN**

Backing up discs with a single disc drive can be achieved by making the source and destination drive numbers identical; the program will alternately ask you to insert the source and destination discs into the drive.

ADFS-DFS

Purpose

Copies files from an ADFS formatted disc onto a DFS formatted disc.

How to use it

Insert the utilities disc into a drive, close the drive door, and select the drive using a `*DIR` or `*MOUNT` instruction. ADFS-DFS is a BASIC program, so to start it type

`CHAIN "ADFS-DFS"` **RETURN**

The `Source Drive` referred to in the question which is now displayed is the drive where the ADFS disc is to be inserted. The drive number to be typed in must correspond to the ADFS system of drive numbering (press **RETURN** for a prompt, and press **RETURN** after entering the drive number). The `Dest`(ination) `Drive` referred to in the next question is the drive where the DFS disc is to be inserted. Again, pressing **RETURN** gives a prompt; remember that if a Winchester drive is present the ADFS floppy drive number is 4; if your system contains a Winchester and you wish to copy some ADFS-format information to it, type in 0 as the drive number, and press **RETURN** (note that in this case use of the ADFS `*COPY` command would be more appropriate). You are then asked to specify the file transfer mode. ADFS-DFS allows files to be transferred singly or in groups. Pressing **RETURN** in answer to the displayed question causes a 'help menu' to be displayed. Typing in

`S` **RETURN**

to transfer a single file results in a straightforward procedure which requires no further explanation here. If you have forgotten the name of the file you want to transfer, use the `C` command to display the contents of the DFS disc.

Typing in

`M` **RETURN**

for multiple file transfer results in all files in the specified source directory being transferred to the specified destination directory. Once again, the destination directory must exist for the transfer to take place.

Typing in

`L`

for list mode results in a similar sequence of events to that for multiple transfer mode, except that each filename is displayed in turn and you are given the option of whether to transfer the file or not.

Notes

If you attempt to transfer an ADFS file which has the same name as a file which already exists on the DFS disc, the transfer will not take place.

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

`Q` **RETURN**

in answer to the `Source drive` question.

Operating system or ADFS commands may be typed in in the normal way at a `Source drive`, `Destination drive`, or `Transfer mode` question. The command will be carried out and the question repeated.

When using ADFS-DFS to transfer programs written under ADFS the following points should be borne in mind:

– any ADFS commands included in a program may need to be changed to their DFS equivalents (if these exist), eg a `*DIR` or `*MOUNT` command would have to be changed to `*DRIVE`.

– any OSWORD calls with A=&70, &71, &72, or &73 will not be recognised by DFS.

DFS-ADFS

Purpose

Copies files from a DFS formatted disc (as used on the BBC Microcomputer) onto an ADFS formatted disc, thus making it readable by ADFS.

How to use it

Insert the utilities disc into a drive, close the drive door, and select the drive using a `*DIR` or `*MOUNT` instruction. DFS-ADFS is a BASIC program, so to start it type

`CHAIN "DFS-ADFS"` **RETURN**

The `Source Drive` referred to in the question which is now displayed is the drive where the DFS disc is to be inserted. The drive number to be typed in must correspond to the DFS system of drive numbering (press **RETURN** for a prompt, and press **RETURN** after entering the drive number). The `Dest`(ination) `Drive` referred to in the next question is the drive where the ADFS disc is to be inserted. Again, pressing **RETURN** gives a prompt; remember that if a Winchester drive is present the ADFS floppy drive number is 4; if your system contains a Winchester and you wish to copy some DFS-format information to it, type in 0 as the drive number, and press **RETURN**. You are then asked to specify the file transfer mode. DFS-ADFS allows files to be transferred singly or in groups. Pressing **RETURN** in answer to the displayed question causes a help menu to be displayed. Typing in

`S` **RETURN**

to transfer a single file results in a straightforward procedure which requires no further explanation here. Pressing **RETURN** in answer to the `Destination Filename` question results in the destination file having the same name as the source file. If you've forgotten the name of the file you want to transfer, use the `C` command to display the contents of the DFS disc. For example, if you attempted to transfer DFS file A.PROG1 to an ADFS disc where there is no directory A, a

```
Not found
```

error message would be displayed and the transfer would not take place. Alternatively, you could give the ADFS copy of the file a different name, and rename it later. See below for further notes on transferring programs to an ADFS environment.

Typing in

`M` **RETURN**

for multiple file transfer results in all files in the specified source directory being transferred to the specified destination directory. Once again, the destination directory must exist for the transfer to take place.

Typing in

`L` **RETURN**

for list mode results in a similar sequence of events to that for multiple transfer mode, except that each filename is displayed in turn and you are given the option of whether to transfer the file or not.

Notes

If you attempt to transfer a DFS file which has the same name as a file which already exists on the ADFS disc, the transfer will not take place.

If

`*`

is typed in as the source directory, all files on the disc will be transferred.

If a file on the DFS disc is in a directory, a directory of the same name will be created on the ADFS disc and the DFS file copied into it. (If such a directory already exists, the DFS file will be copied into it.)

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

`Q` **RETURN**

in answer to the `Source drive` question.

Operating system or ADFS commands may be typed in in the normal way at a `Source drive`, `Destination drive`, or `Transfer mode` question. The command will be carried out and the question repeated.

When using DFS-ADFS to transfer programs written under DFS (ie the DFS floppy disc system), the following points should be borne in mind:

– 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.

– if a program contains instructions which include directory names, the ADFS disc must be given an appropriate directory structure – or the appropriate instructions must be changed. For example, if we have a program called `A.PROG1`, whose sole instruction consists of

```
10 CHAIN "B.PROG1"
```

For `A.PROG1` to run under ADFS either:

(i) directories `$.A` and `$.A.B` must be created on the ADFS disc,

or

(ii) `A.PROG1` could be renamed `PROG1`, `B.PROG1` could be renamed `PROG2`, and line 10 changed to read

```
10 CHAIN "PROG2"
```

– any DFS commands included in a program may need to be changed to their ADFS equivalents (if these exist), eg a `*DRIVE` command would have to be changed to `*DIR` or `*MOUNT`.

– 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).

DIRCOPY

Purpose

Copies the entire contents of a specified source directory (including sub-directories) to a specified destination directory.

How to use it

`DIRCOPY` is a BASIC program, so to start it, type:

`CHAIN "DIRCOPY"` **RETURN**

The source and destination drives will be asked for. If you only have one disc drive, the reply will be the same (0) and you will then be asked if you wish to be prompted to change discs. Answer `Y` if a copy of the source directory is to be made on a disc other than the source disc.

Files in ADFS can be protected against being overwritten by setting the access attribute `L` (locked). This attribute can be ignored on destination files if you answer `Y` to the `Overwrite locked files (Y/N) ?` prompt.

For a source file to be copied, the `R` attribute must be set. If it is not set, a message

```
Can't read file
```

will appear and the file will be ignored.

Since `DIRCOPY` is a BASIC program, it will destroy any program previously in memory.

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, <drv> is the drive number, <sector number> is the sector number (in hexadecimal) where the error was discovered (if appropriate). Some error codes are:

48	– Cyclic redundancy check error
50	– Sector not found
60	– Bad command
61	– Bad address
63	– Volume error
65	– Bad drive

(The error codes are hexadecimal values, the same as would be returned by an OSWORD &72 call). Errors 48 and 50 are not recoverable – the disc must be reformatted if possible. The other errors are all traceable to errors in the command block of an OSWORD &72 call – in particular, errors 61 and 63 indicate that an attempt was made to read off the end of the disc.

&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.

&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 `*COMPACT`ed, 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.

&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
&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
&BD	189	Access violation
&C0	192	Too many open files
&C1	193	Not open for update
&C2	194	Already open
&C3	195	Locked
&C4	196	Already exists
&C6	198	Disc full
&C7	199	Disc error
&CA	202	Data lost, channel
&CB	203	Bad opt
&CC	204	Bad name
&D6	214	Not found
&DE	222	Channel
&DF	223	EOF
&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 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.

1200	Master sequence number
1201 1204	Text to identify the directory
1205 121E	First directory entry
121F	Second directory entry
	End of last directory entry
	0 | Last entry marker
	('garbage')
16CB	0 | Last entry marker (dummy)

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.
`*EXEC`	`*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 ADFS. The details of fitting ROMs in your computer depend on whether you have a Model B or Model B+; the following covers installation in either. Care should be taken to read all the directions which apply to your machine. 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.

A CHARGE MAY BE LEVIED BY THE DEALER FOR FITTING THE ROM TO YOUR BBC MICROCOMPUTER; SUCH A CHARGE SHALL BE ENTIRELY AT THE DISCRETION OF THE DEALER CONCERNED.

Gaining access – BBC Microcomputer Model B+

1. To get to the board, ensure that the computer is disconnected from the mains supply and undo the four large 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. Locate the group of eight large sockets, one or more of which will already contain ROMs (see Fig 8 below). The socket at the rear right of the group (identified on the board as IC71) contains the Operating System/BASIC ROM. The five remaining sockets in the rear group of six (IC62, IC68, IVC35, IC44, IC57) are sideways ROM sockets. The two other large sockets in front of the group of six (identified as IC29 and IC37) are for speech system chips and not for sideways ROMs. ROMs will not work and are likely to be damaged if inserted in these sockets.

Read the section below about the operating priority of the sideways ROM sockets in the BBC Microcomputer model B+ and then insert the ROM as described in the section entitled 'Inserting a ROM'.

Gaining access – BBC Microcomputer Model B

1. To get to the board, ensure that the computer is disconnected from the mains supply and undo the four large 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, release 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 Fig 9. 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 front right-hand side of the main circuit board are fully exposed, then rest it on the lower casing.

4. Locate the row of five large sockets, two or more of which will already be occupied by chips. The four right-hand sockets (identified on the board as IC52, IC88, IC100, IC101) are sideways ROM sockets. The fifth from the right is the operating system socket (IC51).

Read the section below about the operating priority of the sideways ROM sockets in the BBC Microcomputer model B and than insert the ROM as described in the section entitled 'Inserting a ROM'.

Sideways ROMs' operating priorities

The sideways ROM sockets have what is known as an 'operating priority'. Essentially, this means that the ROM which has the highest operating priority will be the one which is selected when the machine is switched on, or after a 'hard reset' (**CTRL BREAK**) is performed.

The operating priority applies to language ROMs (such as BASIC) and Filing System ROMs (in this case, ADFS) independently; both the highest operating priority language ROM and the highest operating priority filing system ROM will be selected after switching on or after a hard reset. Thus, the highest-priority language and the highest-priority filing system are each referred to as the 'default'.

Model B+

Each of the sideways ROM sockets is given a priority ranging from 0 to 15, where 15 is the highest priority and 0 the lowest. As supplied, the priorities of the sideways ROM sockets are as follows: IC71 – 15, IC68 – 11, IC62 – 7, IC44 – 5, IC35 – 3, with the combined Operating System/BASIC ROM inserted in socket IC71.

If you wish ADFS to be your default filing system, the ADFS ROM should be fitted in a socket with a HIGHER operating priority than any other filing system ROM already fitted. For example, if you already have a DFS ROM inserted in, say, socket IC35, the ADFS ROM should be inserted into any socket number HIGHER than IC35. Note that it does not matter if a language ROM is inserted between the two filing system ROMs.

If you do not wish ADFS to be the default filing system, the ADFS ROM may be inserted into any spare socket number LOWER than the one containing your chosen default filing system ROM. If the socket where you wish to insert the ADFS ROM already contains a ROM, that ROM should be removed and inserted into another free socket, though care should be taken not to disturb the operating priority of any language ROMs.

Model B

Examine the five ROM sockets. The one on the left contains the operating system. The four sideways ROM sockets are to right of this socket. The default language ROM (such as BASIC) should be inserted in the right-most socket occupied by a language ROM.

Similarly, if you wish ADFS to be your default filing system, the ADFS ROM should be inserted in a socket to the RIGHT of any socket already occupied by a filing system ROM. Note that it does not matter if a language ROM is inserted between the two filing system ROMs.

If you do not wish ADFS to be your default filing system, then the ROM may be inserted in any socket to the LEFT of your chosen default filing system ROM. If the socket where you wish to insert the ADFS ROM already contains a ROM, that ROM should be removed and inserted in another free socket, though care should be taken to ensure that the operating priority of any language ROMs is not disturbed (see 'Removing a ROM', below).

Removing a ROM

To avoid bending any of its pins, the ROM should be removed very carefully. Take a flat-bladed screwdriver or similar tool and gently prise up each end of the ROM away from its socket, a bit at a time. Be careful not to damage the socket or the circuit board tracks around or underneath it.

Inserting a ROM

1. Having removed the ROM from its protective foam packaging, first check to see if the legs of the IC are parallel with each other. If they appear to be crooked or splayed apart, they should be realigned. To do this, hold the ends of the ROM

between thumb and forefinger and hold one row of legs against a hard, flat surface. Gently press against the hard surface so that the legs bend inwards, a little at a time, until they are at right-angles to the ROM body. Repeat this for the other row of legs until the two rows are parallel with each other.

2. 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.

3. 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.

Fig 8 Inserting ROMs – BBC Microcomputer Model B+

This diagram shows a plan view of the BBC Microcomputer Model B+. The top of the computer casing has been removed to reveal the five sideways ROM sockets. ROM software may be inserted in any of these sockets.

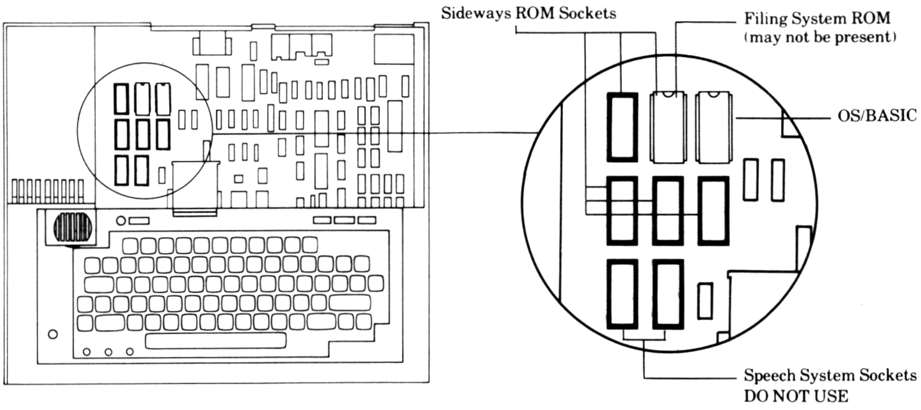
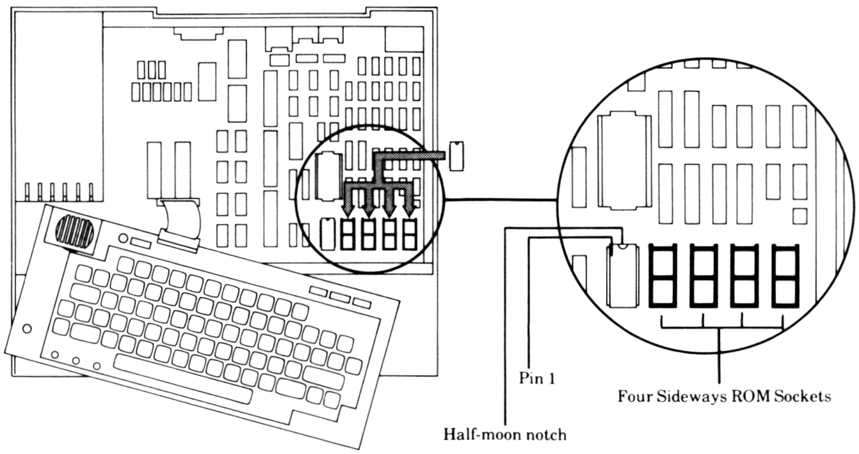

Sideways ROM Sockets
Filing System ROM (may not be present)
OS/BASIC
Speech System Sockets DO NOT USE

Fig 9 Inserting ROMs – BBC Microcomputer Model B

This diagram shows a plan view of the BBC Microcomputer Model B. The top of the computer casing has been removed to reveal the four sideways ROM sockets. ROM software may be inserted in any of these sockets.



Index

ACORN
COMPUTER

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