Advanced Electron DFS

Computer

Products

1770 DFS
USER GUIDE

NOTICE

Advanced Computer Products Limited reserves the right to make improvements to the product described in this manual at any time and without notice.

Advanced Computer Products Ltd,
6 Ava House,
High Street,
CHOBHAM,
Surrey GU24 8LZ.

Tel. (0276) 76545

Advanced Computer Products Limited cannot be held responsible for any loss of data or damage to equipment as a result of this product.

Tube, Electron and Econet are tradenames of Acorn Computers Limited.

CONTENTS

1 GETTING GOING

After fitting the 1770 DFS into your computer, turn on the power. The name of the current filing sytem should appear near the top of the screen. This might be:

ACP 1770 DFS

if 1770 DFS is the current filing system, or

Acorn ADFS

if ADFS is the current filing system.

If neither of these two messages appear then another filing system has been selected.

The current filing system which is selected when you turn on the computer is the filing system which has the higher priority. The ROM socket number of the filing system determines priority. The higher the number, the higher the priority. ROM 15 has the highest priority. So if 1770 DFS is in a higher ROM number than ADFS then 1770 DFS will be selected when the computer is turned on, and vice versa. The DFS command ***ROMS** will tell you which ROM number 1770 DFS is in.

Once a filing system has been selected it does not mean you have to stay in that system. There are three ways you could enter another filing system:

1 By typing the filing system name as a * command e.g.

   ***DISC** to select DFS
   ***ADFS** to select ADFS
   ***TAPE** to select TAPE

2 By resetting the computer and filing system, e.g. by pressing:

   D **BREAK** to select DFS.
   A **BREAK** to select ADFS.
   T **BREAK** to select TAPE.

3 By resetting the computer and initiating an auto-start, e.g. by pressing:

   SHIFT D **BREAK** to auto-boot !BOOT on DFS.
   SHIFT A **BREAK** to auto-boot !BOOT on ADFS.
   SHIFT T **BREAK** selects TAPE only.

To press a key with **BREAK** hold down the keys in the order they are given and release the **BREAK** key first, still holding the other keys down.

2 DISC FILES

Probably the first thing you will want to do with the filing system is to record one of your programs onto a disc. You can do this simply by using the **SAVE** command in BASIC and the filing system takes care of the rest. (NB Not to be confused with ***SAVE** described later in this manual). When you have typed the program into the computer the **SAVE** command causes it to be copied on to the disc. When **SAVE'd**, the program must be given a name. This is called the filename and is used to refer to the program if you later want to copy if back from the disc. Each program **SAVE'd** on to the same drive must be given a unique name. The format of the **SAVE** command is

SAVE 'filename'

where 'filename' can be up to 7 characters. Letters and digits are allowed. The characters

 # * . :

have special meanings which are explained later.

The filename is written into the catalogue together with the sector number where the information starts. Next time you refer to the filename the filing system checks the catalogue to see where the information has been placed on the disc, the old file is deleted and replaced by the new one. The filing system ensures that each new file begins with a new sector.

File specifications

The full specification for a file is

:Drive number.Directory.Filename
:<drive>.<dir>.<filename>

e.g:
:1.Z.MYPROG1

Notice the drive number, directory and filename are separated by full-stops. These are needed so that the computer can distinguish the separate parts of the file specification.

Drive numbers

Drive numbers must be in the range 0 to 3 and preceded by a : (colon). The colon in effect tells the computer: 'This is the start of a file specification, the drive number follows.'

The drives are numbered as shown below. Notice that each side of a double-sided disc is given a separate drive number.

Single-drive, single sided

Drive 0

Dual-drive, double sided

Drive 0	Drive 1
Drive 2	Drive 3

The effect of including the drive number in the full specification is that **:1.$.MYPROG1** is different from **:2.$.MYPROG1** although the file names are the same, they are on different drives.

Directories

The directory is a single character used to divide the catalogue into independent sections. Files of the same name can be created on the same disc with different directories. Although on the same drive,

:1.$.MYPROG is different to **:1.A.MYPROG**

because the directory is different.

Filenames

The filename can be up to 7 of most of the characters on the keyboard in any combination, except the four previously mentioned. When we need to refer to the complete file specification in future we will use the abbreviation <fsp>.

When the filing system is started by pressing **CTRL BREAK** or **SHIFT BREAK**, the current directory and drive number is always set to drive 0 and directory $. The drive and directory can therefore be omitted from file specifications. They will be assumed to have these values.

Typing, **SAVE "MYPROG"**

will automatically store your program in a file name

:0.$.MYPROG

assuming you have not changed the current drive and directory. (Chapter 3 'The filing system commands' explains how you can change the current drive and directory with the commands ***DRIVE** and ***DIR**)

Multi-file operations

Another common term used to refer to multi-file operations is 'Wildcard' facilities. Some of the filing system commands can operate on a number of files instead of just one. These are all followed by the abbreviation <afsp> instead of <fsp>. <afsp> stands for 'ambiguous file specification'. *INFO is an example of such a command. It provides information about a named file, e.g:

***INFO :0.$.MYPROG**

will display information about the file named **MYPROG** in directory **$**

on drive 0.

However, it is possible that you want information about a number of files. The 'WILDCARD' facilities enable you to specify several files for the command to operate on. The wildcards are provided by the characters * and # which have special meanings when they appear in the file specification, e.g:

***INFO :0.#.MYPROG**

means; 'Display information about files called **MYPROG** in any directory on drive 0'.

***INFO :0.$.MYPRO#**

means; 'Display information about all files on drive 0 in directory $ with names starting **"MYPRO"** followed by any single character.' e.g.:

MYPROA, MYPROT and **MYPROG** and so on.

The character * means multiple #'s to the end of the field, eg

INFO :0.$.M

will display information about any files on drive 0 and directory $ whose names begin with **M.**

Auto-start facilities

Sometimes it is useful to make a program or a file on one of your discs *LOAD, *RUN or *EXEC automatically when you insert the disc and press BREAK. This can be done using a file named !BOOT. !BOOT is a special filename recognised by the filing system when you start the computer by pressing SHIFT BREAK. If there is a file of specification

:0.$.!BOOT

the filing system will do one of four things according to the OPTION set on the disc using ***OPT 4,n*** see chapter 3.

Option 0: ignores **!BOOT**
Option 1: ***LOAD's !BOOT** into memory
Option 2: ***RUN's !BOOT** as a machine code program not a BASIC program.
Option 3: ***EXEC'S !BOOT**

See Chapter 3 The filing system commands under the section *EXEC for an explanation of Option 3. That section also describes how to use this auto-start facility to make the computer run one of your BASIC programs automatically.

As well as programs, you may wish to store data on the discs. The filing system provides special facilities for storing and retreiving the data quickly and selectively under the control of your programs.

One of the methods is to use a type of file called a 'Random Access

File' - see chapter 4.

Library directory

The disc system enables you to specify one drive/directory as the 'Library'. This will always be set to **:0.$** when you turn on the computer or press **CTRL BREAK**. It can be altered using the filing system command ***LIB**, until the next **CTRL BREAK**. All the utility programs should be located in the library. This is because when you type

*(Utility name)

it is equivalent to typing

***RUN** (Utility name)

where the drive and directory are omitted and will be assumed to be either the current drive/directory or the Library. The filing systems will first search the current drive/directory for the file and then, if it cannot find it there, it searches the Library.

3 THE FILING SYSTEM COMMANDS

The 1770 Disc Filing System is a 12K byte program. BASIC programs are stored on a disc or tape, but the filing system is stored in Read Only Memory (ROM) inside the computer. The filing system controls the reading and writing of information to and from the disc 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 the computer that a filing system command follows. Each command is described under a number of sections with headings as follows:

Command

This is followed by a syntax abbreviation and a few words explaining the derivation of the word.

<drive> = drive
<fsp> = file specification
<dir> = directory
<afsp> = ambiguous file specification

Purpose

A plain English description of what the command does.

Examples

This section gives a few one-line examples of the use of the commands. These examples are only intended to be illustrative.

Description

A description of the command using normal computer jargon.

Associated commands

This section lists commands which have similar functions or are normally used in conjunction with this command.

Demonstration program

If appropriate a short program is included to illustrate use of the filing system command in a BASIC program.

Notes

Particular points to watch for or special applications of the command are covered by additional notes if necessary.

*ACCESS <afsp> (L)

Purpose

To prevent a file from being deleted or overwritten. The command 'locks' or 'unlocks' a file. You cannot delete, overwrite or write to a locked file until you unlock it again. If you load a file which is locked, you will not be able to save it again with the same name. This is because saving a file with the same name as one already on the disc causes the one on the disc to be deleted and replaced with the new file. A locked file cannot be deleted.

Examples

***ACCESS HELLO L**

This locks the file **HELLO**

***ACCESS HELLO**

unlocks it again so that it can be deleted or overwritten.

Description

Sets or un-sets file protection on a named file. It prevents a number of other filing system commands from acting on the file.

Notes

Once locked a file will not be affected by the following commands:

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

If you attempt to use any of these commands on a locked file the message

File locked

is produced.

If you attempt to use ***ACCESS** on a write protected disc the message

Disc read only

is produced.

Important

Locking a file does NOT prevent it from being removed from a disc with ***FORM40** or ***FORM80** or from being overwritten with ***BACKUP**.

*BACKUP <source> <dest>

Purpose

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

Example

```
*ENABLE
*BACKUP 0 1
```

copies all the information on drive 0 onto drive 1.

Description

Sector by sector copy program.

Associated commands

*COPY
*ENABLE

Notes

If *ENABLE is not typed before the command the prompt Go (Y/N) ? is displayed. Press Y to backup the disc. If you give 0 as the source and destination drives, e.g:

```
*BACKUP 0 0
```

the program will alternately ask you to insert the source and destination discs into drive 0. This makes it possible to copy discs even if you only have a single drive. All information previously on the destination disc is overwritten so be careful not to confuse the source and destination discs. If the source disc is blank the destination disc will end up blank as well.

Warning

The contents of memory may be overwritten by this command. If you have a program or some data in memory that you want to keep, save it before you use this command.

*BUILD <fsp>

Purpose

To create a file directly from the keyboard. After typing this command everything else entered will go into the named file. This is useful for creating EXEC files and the !BOOT file described in chapter 2.

Example

*BUILD !BOOT

will cause everything subsequently typed in to be entered into a file called !BOOT.

Line numbers are displayed on the screen to prompt you to enter your text as follows:

```
*BUILD !BOOT
0010 first line of text
0020 second line of text
0030 ESC
```

Pressing ESC on a line by itself terminates a *BUILD command.

Description

Builds a file from the keyboard.

Associated commands

*EXEC
*LIST
*TYPE

*CAT (<drive>)

Purpose

The command displays the catalogue of a disc on the screen, showing all the files present on the disc. <drive> is the number of the drive you want displayed. If <drive> is omitted the current drive is assumed.

Example

```
*CAT 0
PROGRAM (34) FM
Drive: 0              Option: 2 (RUN)
Dir: 0.$              Lib: 0.$

    !BOOT                 HELLO
    SUMS                  TABLE
    TEST                  VECTORS
    ZOMBIE

  A.HELLO L             B.SUMS
```

Note that the heading part of the catalogue shows the drive number, the title of the disc, the currently set auto-start option of the disc (in this case 2 for RUN), and the currently selected library and directory. The files are displayed in alphabetical order reading across the two columns. In the example above there are nine files on the disc. !BOOT to ZOMBIE are in the current directory $. The current directory's files are always listed first. A.HELLO is in directory A. It is also followed by L, meaning, that it is a 'locked' file. (See *ACCESS for an explanation). B.SUMS is in directory B and is not locked.

Description

Displays a disc catalogue.

Associated commands

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

Notes

Permitted values of <drive> are 0,1,2 or 3. Other values will cause the message

Bad drive

to be displayed and you will have to re-enter the command correctly. The top two lines of the catalogue include the disc title, disc option, drive number and current directory.

Important

The catalogue and hence the disc will hold up to a maximum of 31 files.

*CLOSE

Purpose

To close all sequential access files, and 'ensure' them onto 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**

*COMPACT (<drive>)

Purpose

Attempting to SAVE a program or file on to a disc may produce the message 'Disc full' if there is no single space available on the disc big enough for the information. It may be that there is enough space, but is split into several small sections. This command appends all spare space on a disc to the end. When you delete a number of files, the spaces they had occupied will probably be distributed over the disc with current files in between them. ***COMPACT** moves all current files to the 'start' of the disc leaving the spare space in one continuous block at the end. If <drive> is omitted the current drive is assumed.

Example

```
*COMPACT 1
$.HELLO         FF1900 FF8023 00003B 002
$.SUMS          FF1900 FF8023 000098 003
:
:
:
```

As 'compacting' proceeds, all the current files are displayed in the order in which they occur on the disc. The number of free sectors on the disc is displayed after compacting.

Description

Moves all available space on a disc into one continuous block following the current files.

Associated commands

***FREE**
***MAP**
***SAVE** and BASIC's **SAVE** and **OPENIN**

Notes

This facility will only do anything if there is space between the files. There will only be such space if a file has been deleted from between two others.

Warning

This command may overwrite the contents of memory. If you have a program or data in memory that you want to keep, save it before you use this command.

*COPY <source> <dest.> <afsp>

Purpose

To copy a named file from one disc to another.

Example

***COPY 0 1 HELLO**

This copies a file called HELLO in the current directory on drive 0 onto drive 1.

Description

File copy program.

Associated commands

***BACKUP**

Notes

The 'wildcard' facilities may be used to specify a group of files to be copied e.g:

COPY 0 1 #.MY

Copies all files beginning MY irrespective of which directory they are in. Information already on the destination disc is not affected.

Warning

This command may overwrite the contents of memory. If you have a program or data in memory that you want to keep, save it before you use this command.

*DELETE <fsp>

Purpose

To remove a single named file from the catalogue of a disc. The space occupied by the file becomes available for other information. Succeeding file names in the catalogue are shuffled up, but not the files themselves. Once a file is deleted you cannot get it back again.

Example

***DELETE FRED**

removes a file called **FRED** from the current directory on the current disc.

Description

Single file deletion.

Associated commands

***WIPE**
***DESTROY**
***COMPACT**

Notes

If the disc is write-protected the message

Disc read only

is produced. If the file is not found in the directory the message

File not found

is displayed. If the file is locked the message

File locked

appears. Once deleted a file cannot be restored.

*DESTROY <afsp>

Purpose

To remove specified files from the disc in a single action. This command takes the ambiguous file specification so that groups of files can be deleted. When you use this command a list of the files to be deleted is displayed. A single Yes/No question appears at the end of the list offering you the choice to go ahead and delete all the listed files or not. Use this command with care because its effect is not reversible. It will not attempt to remove locked files. (See ***ACCESS**)

Example

```
*ENABLE
*DESTROY *.H*
A.HELLO
$.HELLO

Delete (Y/N) ?
```

If you type **Y** in reply to the question all the named files will be deleted. The message

Deleted

is displayed when the job is done. Typing anything else cancels the command.

Associated commands

***ENABLE**
***WIPE**
***DELETE**

Notes

Once destroyed files cannot be restored.

If ***ENABLE** is not typed before the command the prompt **Go (Y/N) ?** is displayed. Press **Y** to delete the named files, any other key to cancel the command.

*DIR (<dir>)

Purpose

To change the current directory to <dir>. The current directory is always set to '$' when you press **CTRL BREAK**. To save files in a different directory in the catalogue you must use this command to change the current directory to the one you want and then save them.

Example

***DIR A**

This sets the current directory to A for the current drive. You now have access to any files in directory A in the catalogue. Any files now saved using ***SAVE** or BASIC's **SAVE** will be in directory A.

***DIR :1.A**

This sets the current directory to A in drive 1.

Description

Sets the current directory to the argument supplied.

Notes

Directory can be set to any character except

: "

This command does not alter the directories written in the catalogue. It merely states which directory in the catalogue you have access to by default.

*DRIVE <drive> (40) (80)

Purpose

Changes the current drive to <drive>. Any commands which follow will work on <drive> until another drive is specified. If **40** is included in the command then a 40 track disc can be used in an 80 track drive. This is acheived by moving the head in the disc drive twice the normal amount when seeking to a track. To return the disc drive back to reading 80 track discs, or 'normal stepping' use **80** in this command.

Examples

***DRIVE 1**

Sets the current drive to 1 and

***CAT**

will produce a catalogue of drive 1

***CAT 0**

will catalogue drive 0 but the current drive is still drive 1 until you change it back to 0 or press **BREAK**.

***DRIVE 0 40**

Sets the current drive to 0 and allows drive 0 to read 40 track discs (assuming drive 0 is an 80 track disc drive).

***DRIVE 0 80**

Sets the current drive to 0 and allows drive 0 to read discs normally, i.e. an 80 track drive to read 80 track discs and a 40 track drive to read 40 track discs.

Description

Sets the current drive, and permits 40 track discs to be read by 80 track disc drives.

Notes

After issuing a ***DRIVE <drive> 40** command, the disc in drive <drive> can only be read. An attempt to use a command which writes to the disc will produce the message

Disc read only

*DUMP <fsp>

Purpose

Produces a hexadecimal listing of a file on the screen.

Example

***DUMP SUMS**

Description

Hexadecimal screen dump.

Associated commands

***LIST**
***TYPE**

Notes

It is useful to use this command in page mode so that the file is displayed one page at a time on the screen.

CTRL N selects page mode, CTRL O turns it off.

*ENABLE

Purpose

Some of the filing system commands produce irreversible effects. To prevent them from being used accidentally it is necessary to type *ENABLE before they become operational. These commands are:

*BACKUP
*DESTROY
*FORM

If *ENABLE is not used before these commands then the message

Go (Y/N) ?

will be displayed before the command is performed. Press Y to continue with the command or any other key to cancel it.

Notes

*ENABLE must be typed immediately before the command to be enabled. Any *name command typed in between nullifies the *ENABLE.

*EX (<dir>)

Purpose

To display information about the current directory or directory <dir>. It includes details not given by *CAT. The information is the directory, filename, attribute, load address, execution address, length and disc sector address, the same as displayed by *INFO <afsp>.

Examples

```
*EX

$.!BOOT     L 001900 008023 00003B 005
$.MENU      L 001900 001900 000200 003
```

Displays information about the files in the current directory.

```
*EX D

D.HOME        002000 002000 000300 102
D.WORK        002000 002000 000200 100
```

Displays information about the files in directory D.

Description

Displays directory contents information.

Associated commands

*CAT
*INFO

*EXEC <fsp>

Purpose

This command reads byte by byte all the information in a named file as if it was being typed on the keyboard. This is useful when you find that you are repeatedly typing the same sequence of commands. Instead you can build an EXEC file consisting of all these commands and type *EXEC <fsp> each time you want this sequence of commands. *BUILD <fsp> is an associated command used to create an EXEC file.

Example

*EXEC HELLO

takes the contents of file HELLO and reads it one character at a time as if it was being typed at the keyboard.

Description

Executes the contents of a named file by reading each byte as if it were coming from the keyboard.

Associated commands

*BUILD

Notes

One useful application of the *EXEC command is in association with the auto-start facilities described in chapter 2 and in the section on *OPT 4 in this chapter. If you create a !BOOT file containing the BASIC keyword CHAIN followed by the filename of one of your BASIC programs, the effect of pressing SHIFT BREAK will be to automatically load and run the BASIC program.

*FORM 40/80 (<drives>). . .

Purpose

To initialise a new disc for reading and writing. The first parameter is the number of tracks to be formatted onto the disc. A 40 track drive will use 40 track discs, and an 80 track drive will use 80 track discs. <drive> is the drive number which contains the disc to be formatted. <drive> can be repeated in the command to format more than one disc in the same command. If <drive> is omitted from the command you will be asked to type in the drive number of the drive to format.

A track number is printed in hex as each track is formatted. After formatting a track, it is verified for legibility. See *VERIFY.

Examples

```
*FORM80
Format which drive ? 0
Go (Y/N) ? Y
```

Formats an 80 track disc in drive 0.

```
*FORM40
Format which drive ? 0
Go (Y/N) ? Y
```

Formats a 40 track disc in drive 0.

```
*FORM80 0 2
Go (Y/N) ? Y
```

Formats both sides of an 80 track disc in drive 0.

Notes

This command will overwrite the contents of memory. If you have a program or data in memory that you want to keep, save it before you use this command.

Formatting a disc can cause parts of the screen to be overwritten in some screen modes, however this will not effect the formatting command.

*FREE (<drive>)

Purpose

To display the number of free and used files and disc space remaining and used on the disc in drive <drive>, by default the current drive. Free and used files are given in decimal. Disc space is given as sectors in hex and bytes in decimal.

Examples

***FREE**

Displays the free space on the current drive.

***FREE 2**

Displays the free space on drive 2.

Associated commands

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

*HELP (<keyword>)

Purpose

Displays useful information on the screen. In the disc system this consists of a list of the filing system commands or the utilities depending on the <keyword> used. The two keywords which produce a response in the disc filing system are **UTILS** and **DFS**.

Example

```
*HELP DFS
DFS 2.10

ACCESS   <afsp> (L)
BACKUP   <source> <dest.>
CLOSE
COMPACT  (<drive>)
COPY     <source> <dest.> <afsp>
DELETE   <fsp>
DESTROY  <afsp>
DIR      (<dir>)
DRIVE    (<drive>) (40)(80)
ENABLE
EX       (<dir>)
FORM     40/80 (<drive>)...
FREE     (<drive>)
INFO     <afsp>
LIB      <dir>
MAP      (<drive>)
RENAME   <old fsp> <new fsp>
TITLE    <title>
VERIFY   (<drive>)...
WIPE     <afsp>

*HELP UTILS

DFS 2.10
BUILD    <fsp>
DISC
DUMP     <fsp>
LIST     <fsp>
ROMS     (<roms>)
TYPE     <fsp>
```

Notes

*RUN, *SPOOL, *SAVE, *EXEC, and *LOAD are not included in these lists because they are Machine Operating System commands which operate outside the disc filing system. *HELP is a Machine Operating System command.

*INFO <afsp>

Purpose

Displays information about a file or group of files. It includes details not given by a *CAT such as the length of the file and its location. It is displayed in the following order across the screen.

Directory	Filename	Access	Load Address	Execution Address	Length in bytes	Start sector

Example

***INFO A.HELLO**

Displays
A.HELLO L FF1900 FF8023 00003B 003

Description

Displays detailed file information.

Associated commands

*CAT

Notes

If the file is not found on the specified (or assumed) drive and directory the message

File not found

is produced. The command must be re-entered using the correct <afsp>. The wildcard facilities # and * may be used if you want information about a group of files.

*LIB : (<drive>) <dir>

Purpose

Sets the library to the specified drive and directory.

Example

*LIB :1.A

sets the library to drive 1 directory A. After this typing

* <filename>

will search directory A on drive 1 for the named file and if it is found the file will be loaded and executed just as if you had typed.

*RUN :1.A.<filename>

Description

Sets the drive/directory containing the library.

Associated commands

*RUN

Notes

The library can contain files which are utility programs, designed to act on other files e.g. Sorts, Edits and Merges are all common utility programs. It is then possible to say:

*SORT FRED

where SORT is the name of the file in the library and FRED is the name of another file in the current drive and directory. This makes use of the fact that any text after <fsp> is stored in memory and is available to your machine code program for interpretation. A pointer to the start address of this text is available to your program via a call to OSARGS with Y=0, A=1 and X=the address of the 4 byte block in page 0 where the text is stored. To read the text stored at this location you must use a call to OSWORD with A=5.

OSWORD call with A=5.

Read I/O processor memory. This call enables any program to read a byte in the I/O processor no matter in which processor the program is executing. On entry X and Y point to a block of memory as follows

XY LSB of address to be read
XY+1
XY+2
XY+3 MSB of address to be read

On exit the 8 bit byte will be stored in XY+4.

As this routine reads one byte at a time you may need to use repeated calls to it to recover all the text following the <fsp>.

Chapter 5 gives more information about using the disc system from assembler programs.

*LIST <fsp>

Purpose

Displays a text file on the screen with line numbers.

Example

***LIST DATA**

displays the contents of the file called DATA on the screen, line by line with each line numbered.

Description

ASCII list with line numbers.

Associated commands

***TYPE**
***DUMP**

Notes

BASIC is tokenised so listing a BASIC program file will display nonsense (See the User Guide). An **ASCII** text file of a BASIC program can be obtained using the ***SPOOL** command.

Files written with the BASIC keyword **PRINT#** can also be listed with this command.

In page mode the listing will stop after displaying each screen full until you press either the **SHIFT** key to make it continue. **CTRL** N turns page mode on, **CTRL** O turns it off.

*LOAD <fsp>

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.

Example

*LOAD HELLO

Reads the file HELLO into memory starting at location 1900 (hex), which is the load address of the file when it was saved. (See example in *INFO)

*LOAD HELLO 3200

Reads the file HELLO into memory starting at location 3200 (hex). Other examples are

*LOAD "HELLO"
*LOAD "HELLO" 3200

Description

Loads a file into memory.

Associated commands

*SAVE
*RUN

Notes

All the above are valid commands. The quotation marks are optional; but either a pair, or none should be present. The named file must be in the current directory on the current disc. If the file is not found the message

File not found

is produced.

*MAP (<drive>)

Purpose

To list a map of the free space on the disc in drive <drive>, by default the current drive. The map is printed as a list of disc addresses and lengths of free space in sectors, both in hex.

The free space map changes whenever a file is saved or deleted. This causes spaces to form between files, fragmenting the disc space. To eliminate the free spaces use the command ***COMPACT**.

Examples

***MAP**

Lists the free space map on the current drive.

***MAP 1**

Lists the free space map on drive 1.

Associated commands

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

*OPT 1 (n)

Purpose

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

Examples

*OPT 1 1 or *OPT 1,1

enables the messages;

*OPT 1 0 or *OPT 1,0

disables the messages.

Description

Message system to display file information at every access.

Associated commands

*INFO

Notes

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

*OPT 4 (n)

Purpose

Changes the auto-start option of the disc in the currently selected drive. There are four options to chose from 0, 1, 2 or 3. Each option initiates a different action when you press SHIFT BREAK. The computer will either ignore or automatically LOAD, RUN or EXEC a file called !BOOT which must be in directory $ on drive 0.

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 between the command and (n). *OPT40 would produce the message

Bad option

If the disc is write-protected the error message

Disc read only

is produced in response to the *OPT 4 command.

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

File not found

is produced if !BOOT is not found in directory $ on drive 0.

Important

Do not confuse *OPT 4 with BASIC's keyword OPT or *OPT 1. They are completely different.

*RENAME <old fsp> <new fsp>

Purpose

Changes the file name and moves it to another directory if required.

Example

*RENAME SUMS B.MATHS

Assuming that the current directory is $, the file $.SUMS becomes B.MATHS

Description

Renames a file.

Notes

*RENAME :0.$.SUMS :1.B.MATHS

This is not allowed. The file cannot be moved from drive 0 to drive 1 using *RENAME. Only the directory and filename can be changed. If the file does not exist the message

File not found

is diplayed. If the first file is locked

File locked

is displayed. If the disc is write-protected

Disc read only

is displayed. If the <new fsp> has already been used the message

Exists

is displayed.

*ROMS (<rom>). . .

Purpose

To catalogue sideways Rom <rom>, by default Roms 0 to 15. <rom> can be the title of the Rom as an alternative to the Rom number. The catalogue of a Rom shows its Rom socket number, the type of program in the Rom, the title of the Rom and its version number, if it has one. A Rom can be a language shown as (L), a service Rom (like DFS) shown as (S), or both a language and a service Rom shown as (SL).

Examples

***ROMS**

Catalogues Roms 0 to 15.

***ROMS 15**

Catalogues Rom socket 15.

***ROMS BASIC**

Catalogues the Rom socket containing BASIC.

***ROMS 12 13 14 15**

Catalogues Rom sockets 12,13,14 and 15.

*RUN <fsp> (parameters to utility)

Purpose

This command is used to run machine code programs. It loads a file into memory and then jumps to the execution address of that file.

Example

***RUN PROG**

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

Description

Runs a machine code program.

Associated commands

***SAVE**
***LIB** (for an explanation of 'parameters to utility')

Notes

This command will not run a BASIC progam.

Typing *<fsp> is accepted as being ***RUN** <fsp>.

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

*SAVE <fsp> <start> <end> <exec> <reload>

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 onto the disc in the current drive/directory. It is put into a file of the given name. You will mostly use this command to record your machine code programs.

Example

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

SSSS = Start address of memory to be saved
FFFF = Finish address
EEEE = Execution address
RRRR = Reload address
LLLL = length of information

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 the disc is write-protected the message

Disc read only

is produced. If there are already 31 files on the disc the message

Directory full

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

File locked

is produced. If the file already exists but is unlocked it is deleted. Then starting in sector 2 track 0 a gap large enough to hold the new information is searched for. If none is found the message

Disc full

is produced.

If enough space is available, the information is written onto the disc and the filename is entered onto the catalogue in the current directory.

*SPOOL <fsp>

Purpose

Prepares 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** on the disc ready to receive information from the screen.

LIST

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

***SPOOL**

Turn 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

***BUILD**
***EXEC**
***LIST**
***TYPE**

Notes

BBC BASIC is 'tokenised'. This means that the lines which you type in your program 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. Displaying the file using ***LIST** will therefore produce stange results. The example above shows you how to create a file containing your original program text. If you display that file using ***LIST** your program will appear just as you typed it in.

*TITLE <title>

Purpose

Changes the title of the disc in the current drive to the first twelve characters after the command. It fills in with 'nulls' if there are less than twelve characters. Any characters are allowed.

Examples

*TITLE "MY DISC"

This sets the title to "MY DISC" with five 'nulls' added on the end.

*TITLE "A DIFFERENT TITLE"

This changes the title to A DIFFERENT. Anything after the first 12 characters is ignored. The quotation marks are only required if the title includes spaces.

Notes

If the disc is write-protected the message

Disc read only

appears when you try to use this command.

*TYPE <fsp>

Purpose

Displays a text file on the screen without line numbers.

Example

*TYPE HELLO

Description

Screen list of a named file

Associated commands

*LIST
*DUMP

Notes

BASIC programs are not stored on the disc as text files when you SAVE them so this command will display nonsense.

Page mode is selected with CTRL N and turned off by CTRL O.

*VERIFY (<drive>). . .

Purpose

To verify all sectors of the disc in drive <drive> for legibility. If <drive> is omitted from the command a message is displayed for a drive to verify. Press the number of the drive to verify. <drive> can be repeated in the command to verify more than one disc in the same command. A track number is printed in hex as each track is verified.

If verifying a track fails on the first attempt a ? is printed after the track number. This could mean temporary corruption of the disc, caused by dust, lint or other foreign matter adhering to the surface of the disc.

If verifying a track fails after six attempts the command is terminated and a disc fault is printed. This could mean permanent corruption of the disc, caused by physical damage to the surface of the disc.

Examples

***VERIFY**

Verifies the disc in drive 0

***VERIFY 0 1**

Verifies the discs in drive 0 and 1.

Associated commands

***FORM**

*WIPE <afsp>

Purpose

Removes specified files from the catalogue and rearranges the catalogue. Asks for confirmation that each file conforming to the specification is to be deleted.

Example

WIPE *.SU

is a request to delete all files on the current drive beginning with the letters **SU**. As each file is found the filename is displayed like this

A.SUM :

At this point only type "Y" if you want to delete the file. Typing anything else leaves the file intact.

Description

Delete with <afsp> and confirmation per file.

Associated commands

***DESTROY**
***DELETE**

Notes

Once deleted using ***WIPE**, a file cannot be restored. Locked files are not removed. (See ***ACCESS**)

4 RANDOM ACCESS FILES

One of the major advantages of a disc over a cassette tape is that the read-write head of the disc can be moved to a specific place on the disc quickly and accurately. Imagine you have a data file on a 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 allows 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. The pointer points to a particular character in the file. It is the next character on the file to be read or written. In BASIC the pointer is controlled by the keyword **PTR#**. The other keywords in BASIC 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 User Guide. They are:

OPENOUT
OPENIN
PTR#
EXT#
INPUT#
PRINT#
BGET#
BPUT#
EOF#

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

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

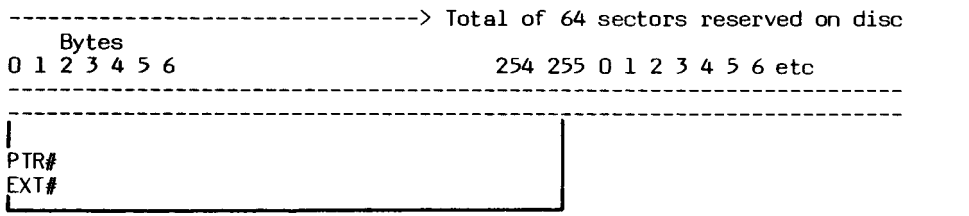
The effect of this line in a BASIC program is as follows:

1 If a file called 'cinemas' exists it is deleted.
2 A file called 'cinemas' is entered on to the disc catalogue of the currently selected drive, in the current directory.
3 The filing system reserves 64 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 64 sectors are not available, the file is not created and an error is produced.
4 Evaluating PTR# and EXT# at this point will reveal that they are both set to zero.
5 The filing system will have loaded into memory the first sector, 256 bytes of the file. This area of memory is specially reserved by the filing system for this purpose and is referred to as the 'Buffer'.

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

If there were no files on the disc previously, the effect can be

illustrated as follows:



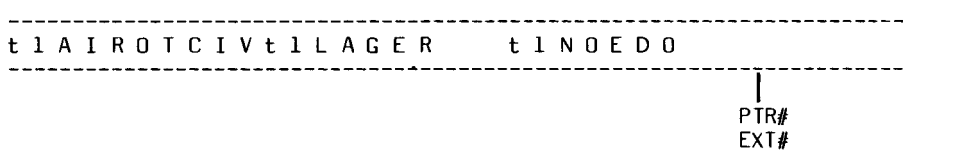
First sector, sector 02 in memory

Nothing has been written on the file, so the value of EXT# (extent) is zero.

We can now use the BASIC keyword PRINT# to write three cinema names into these slots of 10 characters each, as follows:

```
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 (in this illustration, VICTORIA) 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 in the table 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.

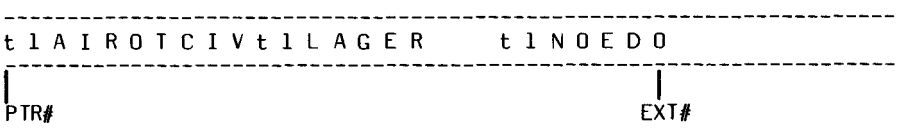
In our example the second byte, represented by 'l', gives the length of the string in hex. The integer and real number types are of fixed length as indicated above so they do not require the byte represented by 'l' to give the length. Real numbers are stored in exponential format, integers are stored 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 02 of the disc and loaded sector 03 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 and 01 are reserved by the disc filing system for the disc catalogue. 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. This empties the buffer and frees the channel on which we opened the file (X in the example).

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

```
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 and 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 name 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.

Note 1

As shown earlier in this discussion, **OPENOUT** reserves 64 sectors for a file. Other files opened may reserve sectors which immediately follow, e.g.

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

The statements reserve 128 sectors consecutively if the disc was otherwise empty.

It may be that you require more than 64 sectors for the first file 'cinemas'. If so, you will need to write 'dummy' records to the file to extend it to the required length before you open the second file. e.g:

```
10 X = OPENOUT("cinemas")
20 FOR A = 1 TO 200
30 PRINT#X, "DUMMY NAME FIELD"
40 PRINT#X, "DUMMY ADDRESS LINE ONE"
50 PRINT#X, "DUMMY ADDRESS LINE TWO"
60 PRINT#X, "ADDRESS LINE THREE"
70 PRINT#X, "POST CODE123"
80 NEXT A
```

This program creates a file 79 sectors long with the dummy name and address written every 100 bytes.

By writing beyond the reserved area in this way you can effectively reserve as many sectors as you like. You can then open other files in the remaining space on the disc. **EXT#** will give the position of the '3' after the last dummy address on the file (20000).

The above method will only work consistently if you start with a blank, formatted disc. If you want to create a random access file larger than 64 sectors on a disc with other files already on it, there is another method.

First save the file with the name you want and with the number of bytes required. Use the address parameters of the ***SAVE** command to specify the number of bytes, e.g:

```
*SAVE "DATA" 0000 08000
```

will create a file of 128 sectors (32k) called DATA. Yuo can then open the file later in your program. The file will contain miscellaneous data which you can overwrite with the information you actually want. This second 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.

Note 2

Up to 5 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.

5 USING THE FILING SYSTEM IN ASSEMBLER

Section 43 of the BBC User Guide or section 29 of the Electron User Guide is essential reading for anyone wanting to write assembler programs. Most of the necessary information for using the filing system in assembler is presented there. In this chapter the main points are summarised and a particular use of OSWORD is described in detail.

General Principles

There are a number of routines available to handle disc I/O. All the routines must be called with a JSR and the decimal flag clear. It is important that you use these routines. They are called in address range &FF00 to &FFFF. They then call an internal (ROM resident) routine whose address is stored in RAM between &0200 and &02FF. The address here will vary according to the filing system in use. For example, the routine **OSFIND** to open or close a file is entered at &FFCE. It is indirected via &021C. &021C and &021D contain the address of the executable routine in the disc filing system ROM. You can intercept the call by changing the addresses in these RAM locations.

Using the available routines you can perform all necessary functions relating to disc files. The relevant routines together with their entry points are as follows:

OSFIND	&FFCE	FINDV	&021C	Open or close a file
OSARGS	&FFDA	ARGSV	&0214	Load or save data about a file
OSFILE	&FFDD	FILEV	&0212	Load or save a complete file
OSBGET	&FFD7	BGETV	&0216	Read a single byte to A from the file
OSBPUT	&FFD4	BPUTV	&0218	Write a single byte to A to the file
OSGBPB	&FFD1	GBPBV	&021A	Load or save a number of bytes
OSWORD	&FFF1	WORDV	&020C	With A=&7F and a parameter block, loads or saves a sector

OSFIND

Opens a file for writing or reading and writing. The routine is entered at &FFCE and indirects via &21C. The value in A determines the type of operation.

A = 0 causes a file to be closed
A = &40 causes a file to be open for input (reading)
A = &80 causes a file to be opened for output (writing)
A = &C0 cuases a file to be opened for input and output (random access)

If A=&40, &80 or &C0 then Y (high byte) and X (low byte) must contain the address of a location in memory which contains the filename terminated with Carriage return (&0D). On exit Y will contain the channel number allocated to the file for all future operations. If Y=0 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 C, N, V and Z are undefined and D=0. The interrupt state is preserved, however interrupts may be enabled during the operation.

OSARGS

This routine enables a file's attributes to be read from file or written to file. The routine is entered at &FFDA and indirects via &214. On entry X must point to your locations in zero page and Y contains the channel number.

If Y is non-zero then A will determine the function to be carried out on the file whose channel number is in Y.

A = 0 read sequential pointer (PTR#)
A = 1 write sequential pointer
A = 2 read length (EXT#)
A = &FF 'ensure' that this file is up to date on the media

If Y is zero then the contents of A will determine the function to be carried out.

A = 0 will return, in A, the type of filing system in use. The value of A on exit has the following significance

0 no filing system
1 1200 baud cassette filing system
2 300 buad cassette filing system
3 ROM filing system
4 Disc filing system
5 Econet filing system
6 Teletext/Prestel Telesoftware filing system
7 IEEE filing system
8 Advanced Disc filing system

A = 1 return address of the rest of the command line in the zero page locations.
A = &FF 'ensure' that all open files are up to date on the media

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.

OSFILE

This routine, by itself, allows a whole file to be loaded or saved. The routine is entered at &FFDD and indirects via &212.

On entry

A indicates the function to be performed. X and Y point to an 18 byte control block anywhere in memory. X contains the low byte of the control block address and Y the high byte. The control block is structured as follows from the base address given by X and Y.

OSFILE control block

00	Address of filename, which must be terminated by &0D	LSB
01		MSB
02	Load address of file	LSB
03		
04		
05		MSB
06	Execution address of file	LSB
07		
08		
09		MSB
0A	Start address of data for write operations,	LSB
0B	or length of the file for read operations	
0C		
0D		MSB
0E	End address of data. that is byte after	LSB
0F	last byte to be written or file attributes	
10		
11		MSB

The table below indicates the function performed by OSFILE for each value of A.

A = 0 Save a section of memory as a named file. The files catalogue information is also written
A = 1 Write the catalogue information for the named file
A = 2 Write the load address (only) for the named file
A = 3 Write the execution (only) for the named file
A = 4 Write the attributes (only) for the named file
A = 5 Read the named files catalogue information. Place the file type in A
A = 6 Delete the named file
A = &FF Load the named file

When loading a file the byte at XY+6 (the LSB of the execution address), determines where the file will be loaded in memory. If it is zero then the file will be loaded to the address given in the control block. If non-zero then the file will be loaded to the address stored with the file when it was created.

The file attributes are stored in four bytes. The least significant 8

bits have the following meanings:

Bit	Meaning
0	not readable by you
1	not writable by you
2	not executable by you
3	not deletable by you
4	not readable by others
5	not writable by others
6	not executable by others
7	not deletable by others

File types returned in A are as follows:

0	nothing found
1	file found
2	directory found

A BRK will occur in the event of an error and this can be trapped if required. 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.

Read/write one byte

OSBGET call address &FFD7 to get a byte
OSBPUT call address &FFD4 to put a byte

Y contains the channel number on which the file was opened using OSFIND.

X is not used, but preserved.

A contains the byte to be put or receives the byte which is read.

The position in the file where the action of the GET or PUT takes place is determined by the position of the pointer as set by OSARGS.

On exit

C clear implies a successfully completed transfer.
C set implies end of file reached before completion of transfer.

Read/write a group of bytes

OSGBPB call address &FFD1.
This routine will read or write a byte (or group of bytes) to or from a specified open file. The option to read or write is determined by the value of A. The length of the data and its location are specified in a control block in memory. This method is particularly useful in the environment of the ECONET filing system where the 'packaging overhead' for transferring small amounts of data is proportionately high.

On entry X (lo-byte) and Y (hi-byte) point to the instruction block:

Offset		
00	File handle	
01	Pointer to data in either I/O processor or second	LSB
02	processor	
03		
04		MSB
05	Number of bytes to transfer	LSB
06		
07		
08		MSB
09	Sequential pointer value to be used for transfer if	LSB
0A	used	
0B		
0C		MSB

A determines the type of operation:

A = 1 Put bytes to media, using the new sequential pointer
A = 2 Put bytes to media, ignoring the new sequential pointer
A = 3 Get bytes from media, using the new sequential pointer
A = 4 Get bytes from media, ignoring the new sequential pointer
A = 5 Get media title, and auto-start option.
The data returned is:
Length of the title (1 byte).
Title in ASCII.
Auto-start option (1 byte).
A = 6 Read the currently selected directory, and device.
The data returned is:
Length of device identity (1 byte).
Device identity in ASCII.
Length of directory name (1 byte).
Directory name in ASCII.
The device identity is the name of the device containing the directory. On the disc filing system it is the drive number, but is not applicable on the ECONET filing system, so its length is zero.
A = 7 Read the currently selected library, and device. The data format is the same as that used for A=6.
A = 8 Read file names from the current directory. The control block is modified, so that the file handle byte contains the 'cycle number', and the sequential pointer is adjusted to ensure that the next call with A=8 gets the next file name. On entry, the number of bytes to transfer is interpreted as the number of file names to transfer; for the first call, the sequential pointer should be zero.
The data returned is:
Length of filename 1

Filename 1
Length of filename 2
Filename 2
etc.

On exit

C clear implies a successfully completed transfer. N, V, and Z are undefined. Interrupt state is preserved, but may be enabled during operation.

The requested transfer cannot be completed if the end of the file has been reached, or there are no more file names to be transferred. In this case, the C flag is set on exit. If a transfer has not been completed the number of bytes or names which have not been transferred are written to the parameter block in the 'number of bytes to transfer field', XY+5. The address field is always adjusted to point to the next byte to be transferred, and the sequential pointer always points to the next entry in the file to be transferred.

Filing system control functions

OSFSC has no direct call address. It is indirected through &21E. It is used for miscellaneous filing system control actions.

A determines the type of operation:

A = 0 A ***OPT** command has been used, X and Y are the two parameters.
A = 1 EOF is being checked. On entry X is the file handle of the file being checked. On exit, X=&FF if an end of file condition exists, X=0 otherwise.

A = 2 A */* command has been used. The '/' character is not part of the command name. The filing system should attempt to ***RUN** the file whose name follows the '/' character.
A = 3 An unrecognised operating system command has been used, the current filing system is offered the command last, after all paged ROMs. The filing system should normally attempt to ***RUN** unrecognised commands. On entry X and Y point to the command name.
A = 4 A ***RUN** command has been used. Load and execute the file whose name is pointed to by X and Y.
A = 5 A ***CAT** command has been used. Produce a catalogue. X and Y point to the rest of the command line.
A = 6 A new filing system is about to take over, so shut down gracefully, close the ***SPOOL** and ***EXEC** files, and do anything else considered necessary.
A = 7 A request has been issued for the range of file handles usable by the filing system. Return in X the the lowest handle issued, and in Y the highest handle possible.
A = 8 This call is issued by the operating system each time it is about to process an operating system command. It is used by the disc system to implement a protection mechanism on dangerous commands, by insisting that the pevious command was ***ENABLE**.

On exit

All registers are undefined, where not defined as described above. Interrupt state is preserved, but may be enabled during operation.

Read/write a sector

OSWORD with A=&7F
Call address at &FFF1.
A=&7F indicates that a general read/write operation is required.

On entry X (lo-byte) and Y (hi-byte) point to the instruction block:

Offset

00	Drive number	
01	Start address in memory of source or destination	LSB
02	address of the data	
03		
04		MSB
05	Number of parameters	
06	Command	
07	Parameters	
08	:	
09	:	

Example:

Number of parameters = 3

Command = &53 to read or &48 to write

Parameter 1 = Track number
Parameter 2 = Sector number
Parameter 3 = &21 (specifies sector length of 256 bytes and 1 to be acted upon)

On exit

0 in the last parameter address + 1 indicates a successful transfer. A failure is indicated by a disc error number.

6 CHANGING FILING SYSTEM

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

*TAPE3	300 baud cassette
*TAPE12	1200 baud cassette
*TAPE	1200 baud cassette
*NET	Econet filing system
*TELESOFT	The prestel and teletext filing system
*ROM	The cartridge ROM filing system
*DISC	Disc filing system
*DISK	alternative spelling for above
*IEEE	IEEE filing system
*ADFS	Advanced disc filing system

7 ERROR MESSAGES

&CC Bad filename
This message appears if you enter a filename which is invalid, such as; longer than 7 characters, etc.

&D6 File not found
The Disc filing system could not find the named file in the specified drive/directories.

&C3 File locked
Access to the named file is locked. This error message is displayed when any attempt is made to overwrite or write to a locked file.

&C8 Disc changed
This error occurs if the computer detects that a disc has been changed while files on it are still open.

&CD Bad drive
This error means that the :(drive) part of the file specification was incorrect, e.g. ':' colon missing or drive number out of range 0 to 3.

&FE Bad command
This means that * was omitted or that the command name was not recognised as a Disc filing system command or utility.

&CE Bad directory
Means that the specified directory is not allowed, e.g. More than one character.

&CF Bad attribute
This error occurs if you use anything other than the letter 'L' with the *ACCESS command.

&CB Bad option
There are currently two 'option' commands *OPT1 and *OPT4. The error occurs if you type anything else besides 1 and 4 after *OPT.

&C6 Disc full
This indicates that there is not enough space on the disc to open (OPENOUT#) or save a file of the specified size.

&BE Catalogue full
The catalogue has enough space for 31 files. This error is produced if you attempt to enter more than 31.

&C4 File exists
This occurs if you try to rename a file with an existing filename.

&C9 Disc read only
This error occurs if you attempt to write to a disc which has the write-protection notch covered.

&C1 File read only
This error occurs if you try to write to a file opened for reading only, using OSFIND with A=&40.

&C7 Disc error NN at DD TT SS
If this error message occurs it means that the computer cannot read the disc. It implies that the disc is damaged, faulty, unformatted or of the wrong type, like an 40 track disc in a 80 track drive. NN is the error number, DD the drive number, TT the track number and SS the sector number.

&C0 Too many open files
This error occurs on attempting to open a sixth random access file. Five is the maximum allowed at once.

&CF Can't extend
An attempt is made to extend a random access file when there is insufficient space immediately after it to do so.

&C2 File open
This error occurs if you attempt to open a file which is already open. An intervening CLOSE is required between two OPENS referring to the same file. This error is also produced if you try to delete an open file with the command ***DELETE**, ***SAVE**, ***BUILD** etc.

8 TECHNICAL INFORMATION

18 bit addressing

The 1770 disc filing system uses 18 bit addressing giving a range from &00000 to &3FFFF. This means that the two bytes and two bits are required to store a complete address, like this:

```
&3FFFF is      11          1111   1111    1111   1111    in binary

             High          Middle          Low
             bits (16-17)  bits (8-15)     bits (0-7)
```

Each full address therefore consists of high, middle and low order bits. This is important to note because the bits of the address are not always stored in consecutively in the catalogue. This is clearly shown by the way that the disc catalogue is loaded into memory.

Another inportant factor concerns the use of a second processor. If the top two bits of an address are set, e.g: &3----, the address is assumed to refer to the I/O processor.

Disc catalogue

Sectors 00 and 01 on the disc are reserved for the catalogue. The format of the catalogue is as follows:

Sector 00

```
-------------------------------------------
    00   01   02   03   04   05   06   07

 00   First 8 bytes of 12 byte disc title
 08   First filename  (7 bytes)      Directory (1 byte)
 10   Second filename (7 bytes)      Directory (1 byte)
 18   Third filename  (7 bytes)      Directory (1 byte)
 :
 F8   31st filename   (7 bytes)      Directory (1 byte)
-------------------------------------------
```

Sector 01

```
-------------------------------------------
    00   01   02   03   04   05   06   07

 00  4 bytes of title     C    N    X    Y
 08  Lm   Ll   Em   El   Bm   Bl   hh   A
 10  Second file's addresses
 18  Third file's addresses
 :
 F8  31st file's addresses
-------------------------------------------
```

where,

C = The disc cycle number.

N = The number of catalogue entries multiplied by 8.

X (bits 4,5) = !BOOT start-up option
(bits 0,1) = Disc size in sectors (2 high order bits)
Y = Disc size in sectors (8 low order bits)

Lm = Load address, (bits 8-15)
Ll = Load address, (bits 0-7)
Em = Execution address, (bits 8-15)
El = Execution address, (bits 0-7)
Bm = Length in bytes, (bits 8-15)
Bl = Length in bytes, (bits 0-7)
hh (bits 0,1) = file's start sector (bits 8-9)
hh (bits 2,3) = file's load address (bits 16-17)
hh (bits 4,5) = file's length in bytes (bits 16-17)
hh (bits 6,7) = file's execution address (bits 16-17)
A = file's start sector (bits 0-7)

Note that the first 8 bytes in sectors 00 and 01 contain miscellaneous information about the disc. Each subsequent block of 8 bytes contain information about the files, repeated for up to 31 files. The complete information about file 3 would be found in the fourth block of 8 bytes on sector 00 followed by the fourth block of 8 bytes on sector 01.

File system initialise and !BOOT

On pressing BREAK the MOS (machine operating system) seeks and initialises a filing system. It starts with the service ROM in the highest priority ROM socket, number 15. The first one will be initialised if just BREAK was pressed. If another key is held down while you press BREAK, the first file system which recognises the key will be initialised. If none recognise the key, the CFS (cassette filing system) is initialised.

DFS recognises the letter 'D', ADFS recognises the letters 'A' and 'F'. To select DFS as the filing system press D BREAK, or A BREAK to select ADFS.

9 FILING SYSTEM COMMAND SUMMARY

Command	Purpose
*ACCESS	- Locks or unlocks a file.
*BACKUP	- Copies all information from one disc to another.
*BUILD	- Causes all subsequent keyboard entries to be stored in a named file.
*CAT	- Displays a disc catalogue.
*CLOSE	- Closes all sequential access files that are open.
*COMPACT	- Collects all files on a disc together into one contiguous sequence leaving a single block of free space.
*COPY	- Copies all specified files from one disc to another.
*DELETE	- Removes a single named file from the disc.
*DESTROY	- Removes specified files from a disc in a single action.
*DIR	- Changes the current set directory.
*DRIVE	- Changes the current set drive.
*DUMP	- Produces a hexadecimal listing of a file.
*EX	- Display the files in a specified directory.
*EXEC	- Reads a disc file byte by byte as if the byte were being typed on the keyboard.
*FORM	- initialises a new disc for read and writing.
*FREE	- Displays the amount of free space on a disc.
*HELP	- Displays the file system commands with syntax guidelines.
*INFO	- Displays information about specified files.
*LIB	- Selects the drive/directory for the library.
*LIST	- Displays a text file on the screen with line numbers.
*LOAD	- Reads a file from disc to memory.
*MAP	- Displays a map of the free space on a disc.
*OPT 1	- Switches screen messages which accompany disc accesses on or off.
*OPT 4	- Specifies the auto-start option of a disc, relating to a named file **:0.$.!BOOT**.
*RENAME	- Changes a filename.
*ROMS	- Produces a catalogue of the sideways ROMS.
*RUN	- Runs a machine code program.
*SAVE	- Saves a specified part of memory to the disc.
*SPOOL	- Transfers all text subsequently displayed on the screen into a specified file.
*TITLE	- Changes the title of a disc.
*TYPE	- Displays a text file on screen with line numbers.
*VERIFY	- Checks legibility of every sector on a disc.
*WIPE	- Removes specified files from the disc after confirmation for each file meeting the given specification.

INDEX

ADVANCED COMPUTER PRODUCTS LTD
6 Ava House, High Street, Chobham, Surrey. Tel: (0276) 76545