Disc Doctor

A ROM based utility program for the BBC micro

Computer Concepts

Computer Concepts
Disc Doctor

PLEASE NOTE

IT IS VITAL THAT THE REGISTRATION CARD SUPPLIED WITH DISC DOCTOR IS RETURNED TO US, WITH YOUR NAME AND ADDRESS FILLED IN. THE CARD IS POSTAGE PAID FOR THE U.K. IF FOR ANY REASON A REGISTRATION CARD IS NOT SUPPLIED, YOU MUST CONTACT THE DEALER FROM WHOM THE PACKAGE WAS PURCHASED. THE SERIAL NUMBER ON THE REGISTRATION CARD SHOULD BE PRINTED INSIDE THE MANUAL. YOU MUST QUOTE YOUR SERIAL NUMBER IN ANY CORRESPONDENCE WITH REGARD TO DISC DOCTOR.

DUE TO INCREASING SOFTWARE PIRACY, A REWARD OF £100-£500 IS OFFERED TO ANYONE PROVIDING INFORMATION LEADING TO A SUCCESSFUL LEGAL SETTLEMENT AGAINST ANY DEALER, SCHOOL, INDIVIDUAL, ETC. MAKING COPIES OF THIS OR ANY OTHER COMPUTER CONCEPTS SOFTWARE PACKAGE.

Phototypesetting by Quorum Technical Services Ltd. Cheltenham, from text prepared on the BBC Micro Computer using the WORDWISE Word Processor.

Computer Concepts cannot be held responsible for any loss of data due to the use of DISC DOCTOR.

INTRODUCTION

The DISC DOCTOR ROM contains a whole host of useful utilities, most for use with disc drives, but some for more general applications such as editing function keys, examining memory, etc.

All the commands are used by typing an asterisk * followed by the particular command word itself. Most commands have one or more arguments, some optional and some compulsory. All the commands available in DISC DOCTOR can be seen by typing the command:

*HELP DISC DOCTOR **RETURN**
or the abbreviation: *H.DI. **RETURN**

This will display the *HELP MENU* on the screen, listing the syntax of all the commands and their arguments. This is further explained in greater detail on the next page.

Various conventions are used throughout the manual to ease clarity for the reader. A key which is to be pressed is shown in reverse print, i.e. white characters on black background. For instance to show that the key marked RETURN should be pressed is shown simply as **RETURN** In addition to this convention all command arguments will be shown in italics, and usually between angle brackets < and >. Exactly what command arguments are is explained in the following pages.

Because of the nature of DISC DOCTOR, parts of this manual are quite technical. Explanations are given where possible, but over-detailed explanations can be as annoying to experts as no explanations at all are to beginners. Relative beginners are therefore advised to refer to appendices such as the 'GLOSSARY OF TERMS' for help. If no explanation is given then you may assume that the information can be found in the USER GUIDE, or that an explanation would be too large to be feasibly included. Section 22 will provide some background information about discs for those who require it.

Command words may be entered in upper or lower case with no distinction made between the two. Because some people prefer to spell the word 'DISC' in the American form 'DISK', all commands that incorporate this word can be spelt either way. So the command ***DISCTAPE** can also be entered as ***DISKTAPE** if required.

Many of the commands can be abbreviated in the usual manner; that is, with one or more letters of the command ending with a full stop. The minimum abbreviation required will depend upon the presence of other * commands available in the machine. The abbreviation is the minimum number of letters required to distinguish it from other commands. This makes it pointless to list abbreviations in this manual.

If a command is given with the wrong syntax then an error message will be issued giving the correct syntax. This removes the need to keep referring to the HELP *MENU* or the manual. The appendices at the back of the manual are provided for quick reference if required.

COMMAND ARGUMENTS

The commands in DISC DOCTOR can accept one or more arguments following them. Some commands *MUST* have arguments following, whereas they are optional for others. An argument is shown as optional if enclosed in parentheses ( ) and compulsory if they are omitted, e.g. the syntax for *EDIT is described by:

***EDIT** (<key no.>)

showing that the key number is an optional argument. All arguments are enclosed in angle brackets to separate them from each other.

The names of arguments for commands are abbreviated both on the help menu produced by DISC DOCTOR and in this manual. A full list of abbreviations and their meanings is given in appendix B at the back of this manual. It helps to familiarise yourself with their meanings.

Arguments should follow the command word, separated by a space. Multiple arguments are each separated by spaces. A general command form could be described syntactically as follows:

***COMMAND** argument argument ...

Taking a real example: the *EDIT command can be given a function key number as an argument. So to edit function key 3 the command would be:

***EDIT 3** **RETURN**

where 3 is the argument to the command. Other commands have different numbers and different kinds of arguments. An argument can be either a number or a string of characters. These two types of argument are now described.

NUMERIC ARGUMENTS

When an argument is of a numeric type, such as a memory address, a *default* radix (or 'number base') applies to each command. The default radix will either be decimal or Hexadecimal (base 16). The following is a summary of default radices for numeric arguments.

HEXADECIMAL	DECIMAL
<sta>	*<trk>*
<end>	*<sct>*
<ofs>	*<key no.>*
<adr>	*<drv>*
<dest page>	*<no. trks>*
<src page>	*<stt>*
<ext>	
<src>	
<dest>	

Regardless of the default radix (number base) expected, a Hexadecimal number can be given by preceding it with an ampersand (&) character, e.g. &FF, &00FF, &FFE7, etc. A decimal number can be given by preceding it with the oblique character (/ normally used by BASIC for divide), e.g. /255, /1234. The largest decimal number which may be input is /65535.

In addition to the 'usual' radices Decimal and Hexadecimal, the facility is provided for entering numbers in any base from 2-99 ! though using a base . greater than sixteen would be highly unusual. To enter a number in a different base is simple, just type the radix followed by a colon (:) followed by the number in that base. For instance, to enter a number in binary, type e.g. 2:10111011. This means that a Hexadecimal number can be entered either as &FFE7 or as 16:FFE7. Both are valid. The following examples show how the same number (Decimal 127) can be entered in different bases –

Binary	2:01111111
(Octal)	8:377
(Decimal)	10:127 OR /127
(Hexadecimal)	16:7F OR &7F

Sometimes it is awkward to give an actual number as an argument and it would be much better to use a variable. For these occasions a special facility is implemented to allow the use of 'system variable' - those from @%...**Z%**. Thus, from within a BASIC program it would be perfectly feasible to have the statement:

***EDIT A%**

which would edit whatever key number is currently in the variable **A%**. The variables @%-Z% may be given as arguments whenever a numeric argument is required.

STRING-TYPE ARGUMENTS

A string-type argument is one consisting of a sequence of characters. If spaces are included in the string then it should be enclosed in double quote marks. If the string of characters 'HELLO' were to be entered as an argument then either **HELLO** or "**HELLO**" is acceptable. If the string 'HELLO SIR' were to be entered then only the form "**HELLO SIR**" with quotes around it is acceptable.

If characters above the range ASCII-126 or between ASCII-0 to ASCII-31 are to be included in the string, then a special syntax must be used. This consists of using the *double-bar* character (positioned next to the cursor-left key on the keyboard) on both sides of the ASCII code to be included. The ASCII code may be entered like any other numeric argument, as described in the previous paragraph, i.e. with any radix.

The string "HELLO" followed by a carriage-return character (ASCII-13) would be entered as **HELLO¦13¦** or **HELLO¦&0D¦** (The default radix is Decimal). Similarly, multiple codes must all be enclosed within double-bars, e.g. **HELLO¦13¦¦10¦** would represent the string of characters 'HELLO' followed by carriage-return (ASCII-13 Decimal) followed by line-feed (ASCII-10 Decimal). A series of control codes alone is a valid argument, e.g. **¦13¦¦10¦¦13¦¦10¦**.

The following pages describe the commands in more detail.

*DIS

FUNCTION : DISASSEMBLER

COMMAND SYNTAX : *DIS (<*sta*>) (<*end*>) (<*ofs*>)

Description

The BBC micro computer is based around the 6502 micro processor. This processor has many instructions which control the operation of the computer. These instructions are stored in memory purely as numbers, called *MACHINE CODE*, which are meaningful to the 6502, but not easily understood by humans. The DIS (disassembler) command interprets these numeric instructions and produces a more easily understandable form called *ASSEMBLY LANGUAGE*. Therefore, the DIS command converts machine code into assembly language.

In order to use the DIS command to any advantage it is necessary to understand 6502 assembly language. If you are unfamiliar with assembly language there is little point in reading further about this command. Output from the disassembler will always be sent to the screen but can be sent to a file (on tape or disc) by using the *EXEC command, or to a printer by using the usual **CTRL-B** combination at any time. While the disassembler is running, a range of keys will cause various options to take effect if pressed. These are described under OPTION KEYS after the arguments below.

Arguments

<*start*> START ADDRESS (OPTIONAL). Disassembly will start from this address if specified or, if omitted, at memory address zero as the default. When disassembling from &8000 to &BFFF (the language area) DISC DOCTOR will disassemble the ROM in socket number 15 as long as it is not DISC DOCTOR. This is the socket on the far right of the language ROM area.

<*end*> END ADDRESS (OPTIONAL). Disassembly will end when this address is encountered, or will continue until **ESCAPE** is pressed if no end address is specified.

<*offset*> OFFSET ADDRESS (OPTIONAL). An offset address makes the disassembled code appear as though it is located at the address given as the offset. e.g. if a piece of machine code is located at &1900 onward and an offset address is given as &8000, then the disassembled listing produced will look as if the code is located at &8000 onward.

The offset will alter the addresses of all instructions that use absolute addresses such as 'JMP nn', 'LDA nn,X', 'STA nn,X'. However, addresses will only be changed if they are in the range of the code being disassembled given by the arguments <*start*> and <*end*>. This prevents subroutine calls in the Operating System, for instance, from being altered in any way.

OPTION KEYS:-

Whilst the disassembler is running, pressing various keys will turn on/off related options. The effect of some of these is not immediately visible and can cause slight confusion if the user is unsure of some commands. Only a few of the options will be used regularly, the rest are of occasional and rather specialised usage. The command keys are listed alphabetically for convenience and a summary is given at the end of the list.

CTRL-KEY COMMANDS

CTRL-B - PRINTER ON
If the **CTRL** key and the letter **B** are pressed together during disassembly then output shown on the screen will also be sent to the printer. Note that output on the screen has no colours whilst being duplicated on the printer. This prevents the undesirable effects achieved by sending colour characters to some printers.

CTRL-C - PRINTER OFF
If the **CTRL** key and the letter **C** are pressed together then output will no longer be sent to the printer. This will have no effect if the printer is already off.

CTRL-N - AUTO-PAGING ON
Output will stop after each screen full of information has been displayed. When ready for the next page of output, pressing either **SHIFT** key will allow the next screen to be displayed, and so on.

CTRL-O - AUTO-PAGING OFF
Output will be uninterrupted. This turns off the effect of the above **CTRL-N** command. There is no effect if auto-paging is already off.

CTRL-L - CLEAR SCREEN
The whole screen is cleared, though current options etc. are not altered, and disassembly continues from the same place.

SINGLE - KEY COMMANDS

SPACE - FOLLOW BRANCH/INVERT JUMP OPTION
If the last instruction displayed on the screen was a branch instruction of one kind or another and the **SPACE** is pressed then that branch instruction will be executed and disassembly will continue from the address specified by the branch instruction. Another use of **SPACE** is to invert JUMP options. If the FOLLOW-JUMPS option is currently off and **SPACE** is pressed when the last instruction displayed was a jump of some kind ('JMP nn', 'JMP (nn)', 'JSR nn' or 'RTS') then the jump will be executed. Similarly if the 'FOLLOW-JUMPS' option was 'on' then pressing **SPACE** will prevent jumps from being carried out. This makes it much easier to follow a program without getting stuck in endless loops.

ESCAPE - RETURN TO INPUT/QUIT DISASSEMBLY

Pressing the **ESCAPE** key once whilst disassembling will cause the disassembly to halt and the prompt:

ENTER ADDRESS :

to be displayed.

When the prompt is issued to enter an address, a new start address may be entered (plus any of the other optional arguments <end> or <ofs> in the usual way), and disassembly will start again from there. Alternatively, pressing **RETURN** alone will allow disassembly to continue from where it left off, as if **ESCAPE** had not been pressed. Pressing **ESCAPE** again when the prompt is given will exit the DIS (disassembler) command altogether. Thus, to exit the disassembler, press **ESCAPE** twice in succession.

B - BACK ONE BYTE

Disassembly continues from the current address less one.

C - CONDITIONALLY FOLLOW JUMPS

This mode of operation acts in much the same way as for normal jumps except that 'JSR nn' instructions (subroutine calls) pointing to locations greater than &8000 (language & Operating System area) will be ignored and any 'JMP nn' or 'JMP (nn)' greater than this value will be treated in the same way as 'RTS' instructions, i.e. stopping the undesired following of subroutines in the operating system and the current language ROM. Subroutine calls into the operating system area are quite frequent in programs and following one takes a long time, so following the same one over and over again can make disassembly almost impossible.

D - DATA MODE

Pressing **D** will enter the data mode, in which machine code is not converted to the assembly language (mnemonic) form, but is instead treated as data. The memory address is shown followed by each data byte, first in decimal, then hexadecimal, then as the corresponding ASCII character. Codes which are outside the range of ASCII characters which can be displayed are simply shown as a dot. The opposite of data mode is mnemonic mode entered by pressing **M** (see later in this section).

F - FAST SCROLL

The screen output is displayed continuously as each instruction is interpreted, i.e. much too fast to read. The paging controls **CTRL-N** and **CTRL-O** may be used to pause the output one page at a time, or pressing **CTRL-SHIFT** together will 'hold' the screen display at any point of interest. The **S** key reverts to 'slow' output.

J - FOLLOW JUMPS

If one of the jump instructions ('JMP nn', 'JMP(nn)', 'JSR nn' or 'RTS') is encountered during disassembly and the 'FOLLOW JUMPS' mode is currently in operation, then the disassembly will follow the course of the jump, i.e. disassembly continues at the address specified as the jump destination. For instance, the instruction JMP &7200 when disassembled with 'FOLLOW JUMPS' in operation will cause disassembly to continue from location &7200. If an RTS (subroutine return) instruction is encountered and a subroutine is currently being followed, then disassembly will continue from where it left off before entering the subroutine, i.e. the return is followed too. If no subroutines are being followed then 'FINISHED' will be output, and the user will be given the option to start disassembly somewhere else. When a subroutine call is encountered, the address to which it should return is stored. Owing to lack of storage space, a limit of one hundred nested returns is imposed, after which a further nested JSR will simply be ignored.

Remember that when spooling to a file, no jumps will *be* followed at all, in order to keep the output file in the correct order. See the description of the <fsp> argument at the start of this section.

M - MNEMONIC MODE

Memory is shown as the assembly language instructions. The display shows on each line-address, instruction mnemonic, each byte in that instruction, and finally the ASCII characters of those bytes. (All the numeric values are shown in Hex.) The example below shows the layout, though colour is used to enhance this on the screen:

```
*DIS 1B77 1B7F
1B77    STA&70    85 70       .p
1B79    STA&72    85 72       .r
1B7B    LDA #&30  A9 30       )0
1B7D    STA &71   85 71       .q
1B7F    LDA #&80  A9 80       ).
```

The opposite of mnemonic mode is data mode, in which each byte is assumed to be data and not converted to the instruction mnemonic (see **D** DATA MODE earlier in this section).

P - BACK ONE PAGE

Disassembly continues from a point one page (&30 bytes) back from the current position.

R – RTS

Whilst in 'FOLLOW JUMPS' mode, **R** will cause a return from the latest subroutine followed, so that disassembly continues from the instruction after the point at which the subroutine was called. If no subroutine is being followed then disassembly ends. This should be used if a lengthy subroutine is accidentally followed.

S - SLOW SCROLL

The opposite of **F**, each instruction is only disassembled as a key (e.g. **RETURN**) is pressed.

DISASSEMBLER COMMAND-KEY SUMMARY

* The default conditions of the various options are marked with an asterisk.

CTRL KEYS

CTRL B	Printer ON	**CTRL N**	Auto-paging ON
CTRL C	* Printer OFF	**CTRL O**	* Auto-paging OFF
CTRL L	Clear screen		

OTHERS

SPACE BAR	Follow branches & invert 'jump condition'
ESCAPE	Enter address / quit disassembly
B	Back one byte
C	Conditionally follow jumps
D	Data Mode (opposite of **M**)
F	Fast Scroll (opposite of **S**)
J	Follow Jumps
M*	Mnemonic mode (opposite of **D**)
P	Back one Page
R	Return from subroutine
S*	Slow scroll (opposite of **F**)

EXAMPLES:-

To disassemble from location &F200 onward use the command:

***DIS F200** **RETURN**

To disassemble from location 12345 (decimal) to 23456 (decimal) use the command:

***DIS /12345 /23456** **RETURN**

(Remember to precede decimal numbers with the oblique symbol '/'.)

To disassemble code in memory at &1900 to &2000 as if it were located at address &4000, i.e. using the *offset* argument, use the command:

***DIS 1900 2000 4000** **RETURN**

*DISCTAPE

FUNCTION : DISC TO TAPE TRANSFER

COMMAND SYNTAX : *DISCTAPE <*afsp*> (<*afsp*>) ...

Description

It is often necessary to transfer programs or data files between discs and cassette tape for a variety of reasons. The DISCTAPE command performs the transfer one way, from disc onto tape, while a later command TAPEDISC will transfer data in the opposite direction.

DISCTAPE is useful, for instance, should you wish to put your own programs onto cassette for someone who does not have discs, or you might wish to keep a back-up copy of all your important programs or data files on tape.

DISCTAPE removes the tedium by doing the hard work for you. If there are specific files to be copied then give a list of their names, and leave DISC DOCTOR to transfer them automatically. If you wish the whole disc to be copied then simply give the command ***DISCTAPE *.*** and press **RETURN**.

Filenames may include 'wildcard' characters. That is to say, if all the files "PROG1", "PROG2", "PROG3", ... are to be copied, you could specify just **PROG*** and all filenames starting with the characters "PROG*" would be copied.

If a capital letter **D** is given as the first filename then all subsequent files will be transferred together with their directory character. **WARNING: This command will corrupt any current memory content.**

Arguments

<*afsp*>... Ambiguous file specification (COMPULSORY). At least one ambiguous filename must be given; this could be the *.* wildcard to transfer all files. The upper case letter 'D' given as this first argument will cause the files to be copied together with their directory character.

(<*afsp*>)... Further ambiguous file specifications (OPTIONAL, MULTIPLE). Optionally, further filenames can be specified in addition to the compulsory one.

Examples

To transfer all files from the current drive onto tape, use the command:

DISCTAPE *. **RETURN**

To transfer all files from drive 2 onto tape, together with their directory letters, use the commands:

***DRIVE 2** **RETURN**
DISCTAPE D *. **RETURN**

To transfer the files: PROG1, TEST, TEST1A, FRED to tape, use:

***DISCTAPE PROG1 TEST TEST1A FRED** **RETURN**

To transfer only the files TEST and TEST1A from the list above, use:

DISCTAPE TEST **RETURN**

*DOWNLOAD

FUNCTION : LOAD AND RELOCATE PROGRAM

COMMAND SYNTAX : *DOWNLOAD <*fsp*> (<*adr*>)

Description

The file specified by <*fsp*> is loaded into memory and then moved to the new origin specified by <*adr*>. If any part of the program is within the area &D00 -&1100 then the cassette filing system is selected to avoid memory corruption. Programs which operate within the area reserved for DFS may be corrupted if DFS commands are used; note that any DFS other than Acorn's may render the reserved area totally unusable while the DFS is selected.

The file is loaded initially to the current O.S. High Water Mark, normally reflected by the value of PAGE in BASIC. The address specified as the new origin can be above or below the current value of PAGE, allowing disc programs to be relocated higher in memory. Whether moving a program up or down in memory, the relocation is carried out intelligentlyso that the program remains intact.

Note that after a BASIC program has been loaded in this way it is sometimes necessary to type 'END' **RETURN** to tell BASIC that the program has moved. Some Machine code programs, such as games, MUST be positioned at a specific address in memory. The DOWNLOAD command cannot change this fact, so do not try to run a machine code cassette-based game at &1900. Most cassette-based games can be run with DOWNLOAD by specifying a new origin address of &E00. Alternatively, if no <*adr*> parameter is specified then the program will automatically be positioned at the re-load address. (If the program has been transferred by TAPEDISC the re-load addresses are copied too).

Arguments

<*fsp*> Filename (COMPULSORY). An unambiguous file specification of the file which is to be downloaded.

<*adr*> Address (OPTIONAL). After loading, the program (or data file) will be moved to the new origin specified by the <*adr*> address argument. If no argument is given then a default action is taken. The default origin is the re-load address saved in the file's information block.

Examples

To LOAD a file called "PROG1" and have it moved to address &C00 use:

***DOWNLOAD PROG1 C00 RETURN**

To LOAD a file called "PROG2" and have it moved to its re-load address use:

***DOWNLOAD PROG2 RETURN**

To load a file called "PROG3" from tape so that it can access discs use:

***TAPE RETURN**

***DOWNLOAD PROG3 1900 RETURN**

Note that in the above case a program is being loaded from cassette, and that it *is* being moved UP in memory (in effect uploaded).

*DSEARCH

FUNCTION : SEARCH ON DISC

COMMAND SYNTAX : *DSEARCH <*str*> <*trk*> (<*trk*><*sct*><*drv*>)

Description

This command allows a disc to be searched for any specified string. In the usual way, strings may include numeric codes too. The search is carried out sector by sector, starting at the track number given by the first <*trk*> parameter. Unless a drive is specified the search will be carried out on the current logged-on drive.

If a match occurs, the disc editor is entered with a display of the sector in which the string was found. Pressing the **COPY** key will start a search for the next occurrence of the string, and so on. See the section on the *DZAP command for details of how to use the disc editor when the string is found.

The second set of parameters comprising (<*trk*><*sct*><*drv*>) are not used in normal circumstances. These arguments specify the number of tracks per disc, the number of sectors per track, and the drive to be used. The number of tracks is usually read from the catalogue in track zero. If track zero is damaged, or an unusual number of sectors per track is used then this is not possible, hence the need to specify the information.

Arguments

<*str*> Search string (COMPULSORY). The string which will be searched for. It may include numeric codes for characters in the standard DISC DOCTOR format.

< *trk* > Start track (COMPULSORY). The search will start at this track on the disc and work onwards from there.

* The following three parameters are optional, but all three MUST be given if any one is given.

<*trk*> Number of tracks (OPTIONAL*). Specifies the number of tracks on the disc to be searched.

<*sct*> Number of sectors (OPTIONAL*). Specifies the number of sectors per track on the disc to be searched.

<*drv*> Drive (OPTIONAL*). Used to specify which drive is to be searched.

*DZAP

FUNCTION : INTERACTIVE DISC EDITOR

COMMAND SYNTAX : *DZAP (<*trk*>) (<*trk*><*sct*><*drv*>)

Description

The DZAP utility is an important part of DISC DOCTOR used whenever a disc needs to be edited for any reason. It is very easy and convenient to use in all circumstances from changing one character in a file, to examining discs from completely different computers. Many further uses can be found in the rest of the manual.

One sector of the disc is displayed at a time. The display is in TELETEXT mode (mode 7) so that it is clear to read and colours can be used to enhance wording. Various keys on the keyboard allow the user to move around the displayed sector and change any character(s) necessary, or simply examine it without changes. Cursor keys are used widely, both on their own and in combination with **SHIFT** or **CTRL**. Movements with these keys are detailed after the arguments below and include- up, down, left, or right one character; move to far left/right of screen; move up/down one screen-full; move forward/backward one sector; move forward/backward one track. If the cursor is moved up or down then the screen will scroll very quickly in the opposite direction to bring the appropriate section onto the screen. This is the same as scrolling in WORDWISE.

A copy of the sector displayed is held in memory for editing, rather than the much slower method of changing the disc directly. If any changes are made to the displayed sector (accidental or deliberate) then the user will be asked **SAVE (Y/N)?** when he tries to exit the routine (by typing **ESCAPE**) or when he moves onto another sector. A 'yes' reply will go ahead and save the changed sector, whilst a 'no' reply will prevent any changes from being made.

The second set of parameters serves the same purpose as in the previous comand DSEARCH, refer back for brief details. More information on their use is given later in the manual.

Arguments

<*trk*> Start track (OPTIONAL). If no <*trk*> parameter is given then sector zero of track zero is displayed by default. If <*trk*> is given then the first sector in that track will be displayed.

* The following three parameters are optional, but all three MUST be given if any one is given. Their use is explained later in the manual.

<*trk*> Number of tracks (OPTIONAL*). Specifies the number of tracks on the disc to be searched.

<*sct*> Number of sectors (OPTIONAL*). Specifies the number of sectors per track on the disc to be searched.

<*drv*> Drive (OPTIONAL*). Used to specify which drive is to be searched.

COMMAND KEYS

ESCAPE Exit DZAP. If any changes have been made to the current sector then the prompt SAVE (Y/N)? will be given before exiting.

TAB Change base for entry of bytes. Normally the entry of bytes is in ASCII; pressing TAB cycles through Hex, Decimal, Binary, then back to ASCII again.

SMALL MOVEMENTS:-

↑ MOVE UP ONE LINE.
↓ MOVE DOWN ONE LINE.
← MOVE LEFT ONE BYTE. If the cursor is moved off the left-hand end then it will be put on the line above.
→ MOVE RIGHT ONE BYTE. If the cursor is moved off the right-hand edge then it will be put on the line below.

LARGE MOVEMENTS:-

SHIFT ↑ MOVE UP ONE SCREEN (16 lines, &80 bytes).
SHIFT ↓ MOVE DOWN ONE SCREEN (16 lines &80 bytes).
SHIFT → MOVE TO FAR RIGHT.
SHIFT ← MOVE TO FAR LEFT.

MOVING TO A NEW SECTOR OR TRACK:-

CTRL ↑ MOVE BACK ONE TRACK. Moving back before the first track will cause a move to the last track.
CTRL ↓ MOVE FORWARD ONE TRACK. Moving beyond the last track will cause a move to the first track.
CTRL → MOVE FORWARD ONE SECTOR. Moving beyond the last sector in a track will cause a move to the first sector of the following track.
CTRL ← MOVE BACK ONE SECTOR. Moving back before the first sector will cause a move to the last sector of the previous track.
CTRL **space** Restart current sector.

ERRORS:-

Various errors may occur occasionally when a disc read/write is performed by DZAP. These error messages will appear at the top of the screen. Generally, nothing can be done to actually rectify these errors.

ERROR L LOAD ERROR. The sector cannot be loaded.
ERROR I INDEX ERROR.
ERROR S SAVE ERROR. The sector cannot be saved.

*EDIT

FUNCTION : DISPLAY FUNCTION KEY DEFINITION(S) FOR EDITING

COMMAND SYNTAX : *EDIT (<*key no.*>)

Description

After entering a long complicated definition for a function key it would often be useful to edit or copy it to another key. The EDIT command provides a very simple method for doing this. It will display the definition of any particular key, or of all the keys. They are displayed in the format required for re-entry, allowing a definition to be edited and re-entered using the standard cursor and **COPY** keys.

All codes entered into definitions using the double-bar and other special characters are converted back correctly so that they may be re-entered. This makes it particularly useful for editing keys used in WORDWISE, which often include special character sequences.

At any time the definitions of all the keys from 0-15 can be seen simply by using the EDIT command without a key number. The EDIT command can be used from within WORDWISE, BEEBCALC or whatever language required, or even a BASIC program.

Argument

<*key no.*> KEY NUMBER (OPTIONAL). If the key number is omitted then the definitions of all keys from 0-15 are displayed. Otherwise the definition of the key specified by <*key no.*> will be displayed.

Examples

To edit the definition of function key number 2 **(f2)** use the command:

***EDIT 2 RETURN**

which will display the definition of the key. Then edit it as required using the cursor and **COPY** keys.

To display the definitions of all function keys at once use the command:

***EDIT RETURN**

*FIND

FUNCTION : FIND OCCURRENCES OF A STRING IN A BASIC PROGRAM

COMMAND SYNTAX : *FIND <*str*>

Description

The FIND command will list all the line numbers in which a specified string of characters occur. Its use is limited to only BASIC programs. Most often the search required will be for a variable, in which case there is no need for anything special to be entered, e.g. ***FIND A$** would list all the line numbers in which A$ was found. A less common search might be for one of BASICs keywords such as GOTO, REPEAT or REM etc, BASIC Keywords such as these are not stored as words but are tokenised. This means that it is necessary to search for the token code that represents the keyword rather than for the word itself. Tokens consists of single byte codes (which are listed in the *BBC USER GUIDE).* To search for single byte codes one includes the code between double-bar characters as normal.

The easiest way to understand FIND is to look at the examples given below.

Argument

<*str*> SEARCH STRING (COMPULSORY). This may include character or keyword token codes in the standard DISC DOCTOR format, i.e. placed between double-bar characters. If a space is included in the string then the whole string must be enclosed within double quotes ("). This means that if the search string is to include double quotes they must be given as the appropriate character code 34 between double-bar characters.

Examples

All the examples will be based upon the example program below:

```
10  REM example program to demonstrate the *FIND command
20  ABC$="EXAMPLE":XBC$=" PROGRAM"
30  PRINT "THIS IS AN ";ABC$;XBC$
40  REM token for END keyword is &E0
50  END
```

To find all occurrences of the variable ABC$:-
type: ***FIND ABC$** **RETURN**
result: 20,30

To find the string PLE" including the double quote character:-
type: ***FIND PLE¦34¦** **RETURN**
result: 20

To find the BASIC keyword END:-
type**:** ***FIND¦&E0¦** **RETURN**
result: 50

NOTE that the actual word END in line 40 is not found, only the keyword.

*FORM

FUNCTION : FORMAT ALL OR PART OF A DISC

COMMAND SYNTAX : *FORM <*drv*> <*no. trks*> (<*stt*>) (<S>)

Description

The FORM command will format a disc to a specified number of tracks. It is necessary to format a new disc before it can be used (see the section on backgound information for more details if formatting is unfamiliar to you.) Beware that formatting a disc will completely erase any information already on it. Because formatting is so potentially disastrous the prompt **ARE YOU SURE? (Y/N)** is always given before formatting starts. This question is your last chance to change your mind before the disc is erased. Lower case **y** is NOT taken as a YES answer.

In some cases it is necessary to format only part of a disc. For instance if one track or a group of tracks is damaged then just the damaged area alone can be re-formatted without harming any other tracks on the disc. If a part format is performed then the catalogue information at the start of the disc can be left unchanged.

The number of tracks per disc used will depend upon the type of disc drives you have. Usually a disc drive will work with either 40 or 80. Note that formatting a disc with only 40 tracks in an 80 track drive does NOT make it readable by a 40 track drive. (If this is difficult to follow please refer to the background information section for an explanation).

In addition to the ordinary formatting described so far, there is a facility to put a special format on discs. The *SWAP command in DISC DOCTOR (see later) allows the use of 60 filenames instead of just the usual 31. But this is only possible on discs with the special format, so if you intend to use 60 filenames then you MUST use the special format on discs right from the start.

Because all discs must be formatted there are already a large number of format utility programs in existence, especially on disc. Format utilities are NOT all the same. Most of them produce discs which actually operate more slowly than necessary in use. *FORM produces discs which run as fast as is possible on the BBC machine.

When floppy discs have been used a lot they can become unreliable in areas. This is also the case with low quality discs or discs which have been physically damaged. During formatting, the number of the track currently being formatted is displayed, one after another. If a track fails to format correctly then a question mark (?) is printed beside the track number causing the error. Further attempts are then made to format the track. If the track still cannot be formatted after several tries the error message 'Format error' will be printed. A question mark indicates that either the drive or the disc may have a slight problem. A Formal error indicates that either the drive or the disc (usually the disc) has a serious error which is preventing formatting of that track. Something like a scratch on the disc would cause a format error. At the end of formatting the question **Verify? (Y/N)** is asked. Verifying is checking that each sector really has been formatted correctly. It can be done separately with the *VERIFY command (see later).

Arguments

<*drv*> DRIVE NUMBER (COMPULSORY). This specifies the number of the drive in which the disc is to be formatted. Only numbers in the range 0-3 will be accepted.

<no. *trks*> NUMBER OF TRACKS (COMPULSORY). This argument specifies the number of tracks to be formatted, which will usually be 40 or 80. However, it could be a different value if an unusual drive is being used such as a 35 track drive, or it will be different if only part of the disc is being formatted. See the examples below.

<*stt*> START TRACK (OPTIONAL). If only part of the disc is to be formatted then this parameter is given to indicate where on the disc formatting should start. The previous parameter <*no. trks*> would be used to show how many tracks should be formatted beyond the starting track <*stt*>.

<S> SPECIAL FORMAT (OPTIONAL). If this parameter is included then the disc will be initialised with a special format. This special format is required for the *SWAP command (offering 60 filenames) to be used. Details of *SWAP are given later in the manual. It is suggested that the letter **S** be used to signify a special format, though in fact the *FORM command will take **ANY** valid string present as the fourth parameter to signify that a special format should be used. This means that the following commands would all be exactly the same and valid ways to specify a special format:

***FORM 0 80 0 S RETURN**
***FORM 0 80 0 SPECIAL RETURN**
***FORM 0 80 0 "WITH A SPECIAL FORMAT PLEASE" RETURN**

Examples

To format a disc in drive zero with 40 tracks use the command:

***FORM 0 40 RETURN**

To format a disc in drive 2 with 80 tracks use the command:

***FORM 2 80 RETURN**

To format only the 4th and 5th tracks on an 80 track disc in drive 0 use:

***FORM 0 2 4 RETURN**

In fact the parameters would be the same to do this on a 40 track disc.

To format the 3rd track onward on an 80 track disc on drive 0 use the command:

***FORM 0 77 3 RETURN**

To put the special format on an 80 track disc in drive 0 use the command:

***FORM 0 80 0 S RETURN**

*JOIN

FUNCTION : JOIN SEVERAL FILES TOGETHER INTO ONE COMMAND

SYNTAX : JOIN <*fsp*> <*afsp*> (<*afsp*>)...

Description

Using the JOIN command several files can be added together end-to-end in one new file. The files which are joined together are left intact on the disc in addition to the new file created. This would enable two WORDWISE files to be joined together for instance. It is also useful for creating backup files consisting of a whole set of small files under just one filename. The files can be separated again by using the *PARTLOAD command for loading any part of a file (described later in the manual).

Another use of *JOIN arises from the fact that the files joined are left intact after the joining. By joining only one file (this may sound strange) it is possible to make another copy of a file on the same side of the disc. See the example at the end of this section. However, note that file information is not copied - see note on next page.

The name(s) of the file(s) to be joined are specified by an ambiguous file specification, so that *wildcard* characters can be used. See the example at the end of this section.

Because of the way that *JOIN works (using BPUTs and BGETs) the destination file may grow to a size in excess of memory size. So a file of nearly 200K can quite easily be created on an 80 track disc. However, all the files being joined and the destination file must be on the current drive. Remember that once a file has been created by joining others, it can then be joined to another set of files, and so on.

Arguments

<fsp> DESTINATION FILENAME (COMPULSORY). The unambiguous name of the file to be created which will hold all the files being joined. This file can only be created on the current drive. Ensure that this filename is not one of those named as a file to be joined, otherwise a *file open* error message will occur.

<afsp> NAME OF FILE TO BE JOINED (COMPULSORY). At least one ambiguous filename must be given as the first or only file to be joined. If no other filenames follow then the JOIN command will have the effect of producing a copy of the file named by <afsp> which will be called <fsp>. The file <afsp> must be present on the current drive.

<afsp> FURTHER AMBIGUOUS NAMES OF FILES TO BE JOINED (OPTIONAL). All further files to be joined are specified here III onwards. The same rules as above apply for the filenames.

 Continued…

Examples

To make a copy of a file called FRED and call it JONES use the command:

***JOIN JONES FRED RETURN**

To join the files FRED, JIM, JONES and SHIELA into a file called FILES use:

***JOIN FILES FILE1 FILE99 FILE3 RETURN**

To join all the files as above in a simpler manner use a wildcard:

***JOIN NEWONE FILE* RETURN**

NOTE: The file information (i.e. load addresss, execution address, etc.) is not copied to the new file. This can be important if *JOIN is used to reproduce machine code programs; it will also effect BASIC programs if they are loaded with MENU, because they are automatically moved to the re-load address, which would be zero on a joined file.

WARNING This command will overwrite any current memory content. Remember to save whatever you are working on first.

*MENU

FUNCTION : DISPLAY PROGRAM MENU

COMMAND SYNTAX : *MENU (*<drv>*) **OR PRESS BREAK-M**

Description

The MENU command looks at the disc and displays a list of files on that disc. This list or menu consists of an alphabetic character followed by the filename. To load and run any program in the list the user simply select your choice by pressing the correct character. It is not necessary to know if the program should be *RUN, LOADed or CHAINed and therefore it is an ideal way of letting a novice RUN programs from the disc and use the computer without having to know all the correct filing system commands.

*MENU will only display files in a special directory and so it is left to the more experienced user to prepare the disc in the first place. MENU can be started by typing *MENU **RETURN** or more easily by holding down the **M** key and pressing **BREAK** at the same time. MENU will only display files under the '+' and '-' directories. BASIC programs should be given a directory character '+', and machine code programs should be given a directory character '-'. (It is necessary to distinguish between BASIC and machine code programs because they must be run in different ways by the MENU program.)

If MENU were started accidentally then pressing **ESCAPE** or even **BREAK** will exit MENU.

Sometimes a program consists of more than one section. The first program loads and runs, which than loads and runs another section, and may go on to load further sections. These further sub-programs should be given the directory character '='. MENU automatically selects '=' as the current directory after loading the first program. If other directory letters are used then the program will either have to change the directory letter with the *DIR command (part of the DFS) or include the directory letter in all filenames.

Arguments

<drv> DRIVE NUMBER (OPTIONAL). If MENU is started with a *MENU command then a drive number can be given. All operations would then take place on that drive. It is not possible to specify a drive when the MENU is started by pressing **BREAK-M**.

Option Keys

CTRL-Q While the menu is displayed, pressing **CTRL-Q** turn off the sound so that subsequent programs will be silent. Some programs however turn the sound on again as part of the program and so this may not always work.

CTRL-S With special format discs (see SWAP) typing **CTRL-S** while the menu is displayed will swap catalogues and give a new menu.

0, **1**, **2**, **3** Pressing any of these numbers will change drive and display a menu from that drive.

NOTE:- All files are loaded into memory and then moved to their respective re-load address, which can be found from a *INFO command. Beware of programs reproduced by *JOIN which would have a re-load address of zero, and consequently not work without this being changed.

Example

Assume that a disc has a title of "DEMO 1" and that it contains BASIC programs called +.BPROG1, +.BPROG2, =.BPROG2A, +.BPROG3 and machine code programs called -.MCPROG1, and -.MCPROG2. Notice that the BASIC programs have a '+' directory except for =.BPROG2A which will be loaded from + .BPROG2, and therefore should not appear in the menu. Also note that the two machine code programs have a '-' directory.

If the command:

***MENU** **RETURN**

were typed or **M-BREAK** were pressed then the screen display would be similar to that shown below...

EXAMPLE OF MENU SCREEN DISPLAY...

DEMO 1

A BPROG1 B BPROG2

C BPROG3 D MCPROG1

E MCPROG2

If you choose to run the program called BPROG1 then you need only press the A key. Similarly, you would need only press E to run the program called MCPROG2, which happens to be a machine code program, though the inexperienced user need not be aware of this.

Often a BASIC program is loaded and run as just a title program, which then loads and runs a machine code program. There are no problems with any combinations of machine code or BASIC. The BASIC title program would be given a '+' directory, and the machine code sub-program would be given an '=' directory, so that it does not appear in the menu.

Files of any other directory can be on the disc and used as normal. No files except those with directories '+' or '-' will be shown as choices in the MENU.

*MOVE

FUNCTION : MOVE A (BASIC) PROGRAM IN MEMORY COMMAND

SYNTAX : *MOVE (<*dest page*>) (<*src page*>)

Description

The uses of the MOVE command are similar to those of the DOWNLOAD command. MOVE will usually be used to position a BASIC program over the area normally used by the DFS, in order to leave more room above the program for storing variables and data. The DOWNLOAD command will do this automatically after loading the program, so it is actually easier. However the MOVE command can be used to move the program back up again, allowing it to be saved. If a large program is being developed then MOVE will need to be used. The program can be moved down to test it, changed as required, moved back up again and saved as the new version.

If the command *MOVE is then entered without any arguments then the whole program will automatically be moved down to address &E00 and the cassette filing system selected to avoid corruption by the DFS.

To move a program back from the cassette area &E00 it is necessary to specify the page to which it should be moved. As stated, a program is located at &1900 for use with discs, so the command would be:

***MOVE 1900 RETURN**

after which the program would be moved and the disc filing system should be selected using the DFS command *DISC. The program could then be saved back on disc.

After a *MOVE command BASIC will be a little confused about where the program is. Type the BASIC command:

END RETURN

after the MOVE and BASIC will sort out all its internal pointers. The program would still work without typing END, but BASIC will think that there is still as much variable space as there was before the program was moved.

Arguments

<*dest page*> DESTINATION PAGE ADDRESS (OPTIONAL). The program will be moved up or down in memory so that it starts at <*dest page*>. If no destination page is specified then the default destination of &E00 will be used, which will also cause the cassette filing system to be selected. The destination page address will take values just as BASIC'S PAGE pointer does. The value of this argument will be rounded down to the nearest page boundary; so &1999 given as the destination page would be automatically rounded to &1900.

 Continued…

Arguments (continued)

<src page> SOURCE PAGE ADDRESS (OPTIONAL). Giving the source page argument specifies the start address of the program to be moved. If no source address is given then the program is assumed to be at the current value of BASIC'S PAGE. As with the previous argument, the value given will be rounded down to the nearest page boundary. This argument is rarely used.

Examples

To move a program from its current position (whatever it may be) to the page boundary &E00, and automatically select the cassette filing system, no parameters are required. Use the command:

To move a program from its current position (whatever it may be) to the page boundary &E00, and automatically select the cassette filing system, no parameters are required. Use the command:

***MOVE RETURN**
END RETURN

Note that programs using the discs for data files etc. while running cannot be located at &E00, because the DFS requires the use of memory between &D00 and &1900.

To move a program from its current location to &1100 use the command:

***MOVE 1100 RETURN**
END RETURN

Note that moving a program to page boundaries of &1100 or above will NOT cause automatic selection of the cassette filing system. Programs located here can use the DFS functions without fear of program corruption, but data files can not and should not be used.

To move a program which starts at page boundary address &E00 up to a new position at &1900 use the command:

***MOVE 1900 RETURN**
END RETURN

Note that the DFS will not be usable until it has been selected with a further command:

***DISC RETURN**

To move a program located at &4000 to a new position at &2000 use the command:

***MOVE 2000 4000 RETURN**
END RETURN

REMEMBER: Always type the command END after moving a BASIC program in order to reset all internal pointers.

*MSEARCH

FUNCTION : FIND SPECIFIED STRING IN MEMORY COMMAND

SYNTAX : *MSEARCH <*str*> (<*adr*>)

Description

MSEARCH will search for any specified string in memory and, when found, it will enter the MZAP memory editor displaying the area of memory containing the string. In the usual DISC DOCTOR convention a string may contain character codes given as the ASCII code between two double-bars, or it may consist entirely of codes. If spaces are included in the string then the whole string must be enclosed within double-quote marks. See section one for more details on using arguments.

When the string has been found and the memory editor entered, pressing the **COPY** key will cause MSEARCH to find the next occurrence of the string, and so on until the end of memory is reached. For details of how to use the memory editor when entered, see the following section on the MZAP command.

Occasionally it is possible to be fooled into thinking that MSEARCH is not working. Loacations at the start of memory are used by DISC DOCTOR and other utilities/languages to store all sorts of pointers and other information. These locations are constantly changing, so it is possible for MSEARCH to find a match for the specified string which, when the area of memory is displayed, has already changed! There will also be one or more copies of the string held in the utility/language workspace which may be confusing if you find them by accident. Usually there is one copy in the keyboard entry buffer (the position of which is dependent upon the language being used at the time) and there will be a copy used by MSEARCH for the purpose of comparison. The irrelevant copies can be avoided if the search started from PAGE (&E00 or &1900).

Arguments

<*str*> SEARCH STRING (COMPULSORY). The string should be entered according to DISC DOCTOR standards (section 1), and is case specific.

<*adr*> SEARCH START ADDRESS (OPTIONAL). The search will start at the address specified by <*adr*>. If the start address is omitted then the search will start from zero.

Examples

To search for the string of characters 'ABCD' use the command:

***MSEARCH ABCD RETURN**

To search for the string 'Disc Doctor' starting the search at address &1900 use the command:

***MSEARCH "Disc Doctor" 1900 RETURN**

Note that the string inludes a space and must therefore be enclosed within double-quote marks, and that the search carried out is case specific.

*MZAP

FUNCTION : INTERACTIVE MEMORY EDITOR COMMAND

SYNTAX : *MZAP <*adr*>

Description

MZAP is a fully interactive on-screen memory editor, displaying any area of memory and allowing changes to be made quickly and easily. The screen acts as a window into memory, showing just a small area of the total memory. Controls are provided for moving this window to show any part of the memory whatsoever. The memory editor is very similar to the disc editor (DZAP). Those familiar with the disc editor should be able to use MZAP with only a quick reminder of controls on the following page.

Cursor keys are used for moving the display window around memory: cursor keys alone move one byte to the left/right or one line up/down. Cursor keys in conjunction with SHIFT move to the far left/right or up/down 16 lines (&80 bytes). In order to make an alteration, the cursor (shown as two arrows, one on each side of the byte) is moved to the position where a change is to be made and a new value typed directly. When MZAP is first entered, an ASCII default is applied for entering new values. This means that to enter a particular character it is necessary only to press the appropriate key. In ASCII mode the cursor will automatically move onto the next byte, making it easy to enter strings of characters. If bytes are to be entered numerically they can be entered in Decimal, Hexadecimal, or even Binary. The current default for entering numbers is indicated in the top right of the display by one of the letters A,D,H,B as appropriate, and is changed by pressing the **TAB** key.

An example of just part of the MZAP screen display is shown below to give an idea of what can be seen.

2228	50	20	73	63	72	65	65	6E	P screen
2230	20	64	69	73	70	6C	61	79	display
2238	20	69	73	20	73	68	6F	77	is show
2240	6E	20	62	65	6C	6F	77	20	n below
2248	74	6F	20	8D	67	69	76	65	to .give
2250	*20*	61	6E	20	69	64	65	61	an idea
2258	20	6F	66	20	77	68	61	74	of what
2260	20	63	61	6E	20	62	65	20	can be
2268	73	65	65	6E	2E	0D	00	00	seen....
2270	00	00	00	00	00	00	00	00	.....

The cursor is shown (two arrows) pointing to the byte at address &2250.

Argument

<*adr*> MEMORY ADDRESS TO BE EDITED (OPTIONAL). If the <*adr*> argument is omitted then the area at the current value of BASICs PAGE will be displayed. (This may be different in a language other that BASIC.) If <*adr*> is included then the area displayed initially will be that around the specified address.

*MZAP COMMAND KEYS

GENERAL:-

ESCAPE Exit MZAP.

TAB Change base for entry of bytes. Normally the entry of bytes is in ASCII; pressing **TAB** cycles through Hex, Decimal, Binary, then back to ASCII again. The cursor will automatically step-on while in ASCII mode, but not while in numeric entry modes.

SMALL MOVEMENTS:-

↑ MOVE UP ONE LINE. If the cursor moves beyond address zero then the top end of memory comes onto the display.

↓ MOVE DOWN ONE LINE. If the cursor is moved beyond address &FFFF then the bottom end of memory comes onto the display.

← MOVE LEFT ONE BYTE. If the cursor is moved off the left-hand end then it will be put on the line above.

→ MOVE RIGHT ONE BYTE. If the cursor is moved off the right-hand edge then it will be put on the line below.

LARGE MOVEMENTS:-

SHIFT ↑ MOVE UP ONE SCREEN (16 lines, &80 bytes).

SHIFT ↓ MOVE DOWN ONE SCREEN (16 lines, &80 bytes).

SHIFT → MOVE TO FAR RIGHT.

SHIFT ← MOVE TO FAR LEFT.

*PARTLOAD

FUNCTION : LOAD ANY PART OF A LONG FILE

COMMAND SYNTAX : *PARTLOAD <*fsp*> <*ofs*> <*ext*> <*adr*>

Description

Commands that already exist for loading files do not allow for loading just part of a file. The only way to do this is with specially written programs using file handling commands of BASIC or the OPERATING system. The existing *LOAD command will only loaad an entire file, so that a file larger than the free area of user memory cannot be loaded. PARTLOAD will allow any specified section of a file to be loaded. It is necessary to give arguments specifying the name of the file, the starting position in the file of the block to be loaded, how much is to be loaded, and the address in memory where it should be loaded.

Arguments

<*fsp*> UNAMBIGUOUS FILENAME (COMPULSORY). The name of the file from which the part is to be loaded.

<*ofs*> OFFSET FROM START OF FILE (COMPULSORY). The start of the block to be loaded must be specified as an offset from the start of the file. The offset is specified as the number of sectors (256 byte sections), rather than the number of bytes. (This allows larger offsets and therefore larger files can be used.) The file from which the section is being loaded is only limited in size by the disc capacity.

<*ext*> EXTENT OF SECTION TO BE LOADED (COMPULSORY). The extent is the size of the section to be loaded, expressed as a number of bytes.

<*adr*> LOAD ADDRESS (COMPULSORY). The part of the file already specified will be loaded into memory at the address given by the <*adr*> argument.

Example

To load a &100 byte secton of a file, starting &500 bytes (5 sectors) into the file named LARGE1, and place it in memory at location &2000 use the command:

***PARTLOAD LARGE1 5 100 2000 RETURN**

Note that the start of the section is expressed as a number of sectors from the beginning of the file.

*RECOVER

FUNCTION : LOAD SPECIFIED SECTORS INTO MEMORY

COMMAND SYNTAX : *RECOVER <*trk*> <*sct*> <*sct*> <*adr*> <*drv*>

Description

The RECOVER command is used mainly for recovering sectors directly from the surface of a disc. Any number of sectors starting at a specified track and sector on the disc can be loaded into memory. Once in memory the area loaded can be examined/edited with the MZAP memory editor, then saved back to the disc either with *SAVE or with RESTORE (see following section).

Arguments

<*trk*> START TRACK (COMPULSORY). Together with the following <*sct*> argument it specifies the disc address of the first sector to be loaded.

<*sct*> START SECTOR (COMPULSORY). Together with the previous <*trk*> argument it specifies the disc address of the first sector to be loaded.

<*sct*> NUMBER OF SECTORS TO BE LOADED (COMPULSORY). The length of the section to be loaded is specified by <sct> expressed as a number of sectors. This will usually take a little work to calculate. Either use DZAP to decide which sectors are to be loaded or, if you know how many bytes you want to load, then use the following calculation formula:

no. sectors = INT ((No.bytes -1)/ 256) + 1

There are 256 (Decimal) bytes per sector. It may also be useful to remember that there are ten sectors per track. The above formula can be used from BASIC to calculate the number of sectors to be given as the argument (in Decimal).

<*adr*> LOAD ADDRESS (COMPULSORY). The number of sectors specified will be loaded into memory at the address given by the <*adr*> argument.

<*drv*> DRIVE NUMBER (COMPULSORY). The drive from which the sectors are to be loaded must be specified.

Example

To load eleven sectors (&B) from drive zero (the first of those sectors is at track 1 sector 3) and place it in memory at address &2000, use the command:

***RECOVER 1 3 11 2000 0 RETURN**

Then to store that area in NEWFILE on drive two, the *SAVE command could be used:

***SAVE NEWFILE 2000 2B00 RETURN**

Note that if &B sectors (11Decimal) have been loaded then the number of bytes is &B00.

*RESTORE

FUNCTION : SAVE SPECIFIED SECTORS FROM MEMORY ONTO DISC

COMMAND SYNTAX : *RESTORE *<trk> <sct> <sct> <adr> <drv>*

Description

The RESTORE command is used mainly in conjunction with the *RECOVER command (see previous section) for restoring/replacing damaged sectors of a disc. An area of memory can be saved directly onto the disc starting at a specified track and sector. The memory area is saved in complete sectors, so it is not possible to RESTORE half a sector for instance. The entire sector must be set up in memory first (by using MZAP usually) and then saved onto the disc with this command.
Saving and loading to and from the disc directly, ignoring the DFS conventions and catalogue etc. is posssible with *RESTORE and *RECOVER, though not recommended as common practice.

Arguments

<trk> START TRACK (COMPULSORY). Together with the following *<sct>* argument it specifies the disc address where the first sector will be saved.

<sct> START SECTOR (COMPULSORY). Together with the previous *<trk>* argument it specifies the disc address where the first sector will be saved.

<sct> NUMBER OF SECTORS TO BE SAVED (COMPULSORY). The length of the section to be saved is specified by *<sct>* expressed as a number of sectors. If you know how many bytes are to be saved then use the following calculation formula to determine the number of sectors to save:

no.sectors=INT((No.bytes-1) / 256) + 1

There are 256 (Decimal) bytes per sector. Because only complete sectors are saved the above formula assumes that an extra sector is to be saved if there is any remainder. The number of sectors to be saved is expressed in Decimal.

<adr> START ADDRESS OF MEMORY TO BE SAVED (COMPULSORY). The number of sectors specified will be saved from the memory starting at *<adr>*.

<drv> DRIVE NUMBER (COMPULSORY). The drive on which the sectors are to be saved must be specified.

Example

To save eleven sectors (&B) on drive zero starting at track1 sector 3, using the memory image at address &2000 onward, use the command:

***RESTORE 1 3 11 2000 0 RETURN**

Note that the above example would have saved memory between &2000 and &2B00.

*SHIFT

FUNCTION : SHIFT A BLOCK OF MEMORY

COMMAND SYNTAX : *SHIFT *<src> <dest>* ***<ext>***

Description

A block of memory can be moved from one place to any other place in memory using *SHIFT. The source and destination addresses are specified together with the number of bytes to be moved. The shifting operation is carried out 'intelligently' so that the source block is not overwritten by the destination block whether moving up or down in memory, even if the source and destination blocks overlap.

SHIFT would not usually be employed for the purpose of moving BASIC programs, since the *MOVE command exists for moving between page boundaries. The main differences between SHIFT and MOVE are that MOVE will only take source and destination addresses of page boundaries (BASIC programs may only start at a page boundary) whilst SHIFT will take any source or destination address. MOVE will use appropriate defaults for source and destination whilst there are no sensible defaults for SHIFT and, finally, MOVE will find the length of the BASIC program starting at the source address and move that amount, whilst SHIFT must be told the extent.

Arguments

<src> SOURCE ADDRESS (COMPULSORY). Specifies the start address in memory of the block to be moved.

<dest> DESTINATION ADDRESS (COMPULSORY). Specifies the address in memory to which the block must be moved. The destination may be above or below the source, and may overlap the source block if required.

<ext> EXTENT (COMPULSORY). The number of bytes to be moved is specified by *<ext>* .

Examples

To SHIFT a block of &100 (256) bytes from &4000 to &5000 use the command:

***SHIFT 4000 5000 100 RETURN**

To shift a &1000-byte block of memory located at &2000 up by just one byte (i.e.the source and destination overlap) use the command:

***SHIFT 2000 2001 1000 RETURN**

*SWAP

FUNCTION : SWAP TO ALTERNATE CATALOGUE (ALLOWS 60 FILENAMES)

COMMAND SYNTAX : *SWAP (<*drv*>)

Description

By use of the SWAP command it is possible to have up to 60 files per side of a disc. This is done by treating each side as two separate halves, so that an 80 track disc would still have 200K on a side, but it would be in two 100K sections.

This command can only be used on discs that have been specially formatted with the *FORM command by specifying a final (S) parameter (see *FORM, section 9). This means that a disc which already contains information cannot be converted to a special format without consequently erasing everything. Instead, a special format disc should be created and the files copied onto it.

Typing *SWAP with a special format disc will swap the file catalogue for an alternative one which holds the other half of the files. Each of the file catalogues can hold 31 filenames, but one filename is used by SWAP to hold and protect the other half of the files. This leaves 30 filenames per catalogue, making 60 per disc side. The special filename will appear in the catalogue as !.!!!!!!!L and is locked against deleting. This special file MUST NOT ever be deleted, or the first half of the disc will be lost! Although the special file is locked, it can be unlocked with the DFS command *ACCESS*.* **RETURN**. this occurs it MUST be locked immediately to protect it.

Typing *SWAP will exchange file catalogues. Each catalogue can be used in exactly the same way as usual. A title can be given to each of the catalogues individually, so you could distingush them by using the sequence of commands:

***TITLE "Catalogue 1" RETURN**
***SWAP RETURN**
***TITLE "Catalogue2" RETURN**
***SWAP RETURN**

...or anything similar of your choice.

The total capacity of the disc remains unchanged but now with two catalogues (effectively splitting the disc in two) each of which can only access half the normal space. This means that very long files may cause a 'Disc Full' error message sooner than expected. Remember, the capacity of a special format catalogue is 50K on a 40 track disc, or 100K on an 80 track disc.

It is not possible to directly copy between the special catalogues on one side of a disc. If one single-sided drive is being used then files should be *LOADed into memory, the catalogues swapped, then the file *SAVEd into the new catalogue. If another disc side is present then just copy the file to another side, swap cataloges, and copy it back to the new catalogue. There is nothing to prevent copying between any combination of first and second catalogues on different sides of discs. It is possible for instance, to copy from the first catalogue on drive 0 into the second catalogue on drive 2.

The special catalogues are swapped entirely and remain swapped when the disc is removed or the machine is switched off. So when a disc is inserted, the catalogue shown will be that which was current when the disc was last used. If a secial format disc is used on a machine without DISC DOCTOR installed, the current catalogue and all files within it will function perfectly. However, it will not be possible to use the *SWAP command to swap the directories, so that the alternative directory will not be accessible.

When saving sectors onto a disc directly, or using the disc editor, it is always important to avoid damaging the file catalogue area in sectors zero and one on the first track. In addition it is important to avoid damaging the special file named !.!!!!!!! (to make it stand out) which holds the alternative file catalogue on a special format disc.

If an attempt is made to use the *SWAP command on a disc with standard format, the error message 'Not found' will be produced indicating that the alternative catalogue could not be found, i.e. the file !.!!!!!!! does not exist. This means that a special format disc can be distinguished from a standard format by using the *SWAP command. If no error message is given then the disc must be special format.

All the normal DFS commands can be used with special discs with the exception of *BACKUP which has some strange side effects. However if all the files are to be copied from one disc to another then *COPY x y *.* will work just as well.

Argument

<drv> DRIVE NUMBER (OPTIONAL). Specifies the drive on which the catalogue should be swapped. If no argument is given then the current drive is assumed.

*TAPEDISC

FUNCTION : TAPE TO DISC TRANSFER

COMMAND SYNTAX : *TAPEDISC (<*fsp*>)...

Description

Any valid standard file on cassette (programs, data files, etc.) can be transferred to disc. Names of specific files to be transferred may be given, or all files may be copied automatically as they are found until **ESCAPE** is pressed.

Ambiguous filenames may not be given. If errors are found as blocks are loaded, then error messages will be issued and re-try prompts given as usual. If filenames are specified for transfer they must be given in the order in which they are stored on the tape.

Because tape filenames may have ten characters and disc filenames can only have seven characters, filenames may be shortened when copied to disc. If a filename on tape starts with a single character followed by a dot (.) then this will become the directory letter on the disc filename. The following are examples of shortening which may occur if long filenames are used on cassette:

CASSETTE		DISC
A.D-DOCTOR	=	A.D-DOCTO
DISCDOCTOR	=	DISCDOC

Argument(s)

<*fsp*>... FILE SPECIFICATION(S) (OPTIONAL). One or more names of files to be transferred may be given. No wildcards are allowed. Filenames must be given in the order in which they are recorded on the tape (though other files may appear in between). If no filenames are specified then all files are copied as they are found.

Examples

To copy all files from tape to disc use the command:

***TAPEDISC RETURN**

To copy the files named PROG, FILE and PICTURE onto disc use the command:

***TAPEDISC PROG FILE PICTURE RETURN**

*VERIFY

FUNCTION : CHECK READABILITY OF DISC

COMMAND SYNTAX : *VERIFY (<*drv*>) (<*no. trks*>) (<*stt*>)

Description

VERIFY will check all or specified tracks on a disc to ensure that they can be read correctly. Discs which are incorrectly formatted or damaged will be detected.

During verification, the number of the track currently being verified is displayed. If an error is detected then a question mark (?) is printed beside the track number. Nine further attempts will be made to read the bad track (ten attempts in all). If the track is read without error by the tenth attempt then the next track is verified, and so on. But if the track cannot be read within ten attempts the error message 'Verify error' is displayed and verification ceases.

VERIFYING SPECIAL-FORMAT DISCS

Note that when verifying a special-format disc (see sections 9 and 19 on FORM and SWAP), there are half the usual number of tracks per side in ONE of the catalogues. So a *VERIFY command with no parameters may find 80 tracks as expected on an 80 track drive, but after a *SWAP command the *VERIFY would only find 40 tracks. This is perfectly normal on special-format discs, and does not indicate an error.

Argument(s)

(<*drv*>)... DRIVE NUMBER (OPTIONAL). Drive to be verified. If no parameters are given then the current drive is assumed.

(<*no. trks* >) NUMBER OF TRACKS (OPTIONAL). The number of tracks to be verified can be given when only a partial verify is required.

(<*stt*>) START TRACK (OPTIONAL). Verifying will start at the track specified by <*stt*> and continue for <*no. trks*> tracks onward.

Examples

To verify all tracks of a disc (with any number of tracks) which is in the current drive, use the command:

***VERIFY RETURN**

To verify tracks 5, 6 and 7 only of a disc with any number of tracks (greater than 7) which is in drive zero use the command:

***VERIFY 0 3 5 RETURN**

INTRODUCTION TO DISC FORMATS

Information is stored magnetically on a floppy disc in concentric circles called tracks. There are only two different types of disc system available on the BBC machine - a 40 track disc system and the more expensive 80 track system. The only difference is the number of tracks recorded on the surface of the disc and therefore the amount of information that can be stored. The outermost track is referred to as track 00 and the innermost as track 39 (or 79). Each track on the disc holds exactly the same amount of information, namely 2560 bytes, so altogether a 40 track disc holds 102400 bytes per surface - the 80 track disc will hold twice that amount. Some types of disc drives can access both sides of the disc. The most common types of system available for the BBC machine are 40 track single sided and 80 track double sided.

Each track on the disc is split up into equal size blocks of 256 bytes, called sectors. There are always 10 sectors per track making the 2560 bytes per track. As just mentioned, the BBC disc format has 10 sectors per track and either 40 or 80 tracks per side of the disc. Other computer systems use other formats and so it is quite unusual to be able to read discs from other computers. Other disc formats might have only 128 bytes per sector, and differing numbers of sectors per track, or only 35 tracks per side.

Before a disc can be used it must have special marks recorded on it stating where the tracks are, where the sectors start, and the number of sectors per track, etc. This information is written onto a blank disc using a FORMAT program. This is normally supplied separately on disc but an expanded FORMAT routine has been included in DISC DOCTOR. This allows a blank disc to be formatted to any number of tracks (usually 40 or 80). For more details see the *FORM command.

One of the great advantages of a disc filing system over a cassette system is that it is possible to load a file off the disc by just giving its name. It does not matter where or when the file was saved, the disc system simply looks up the necessary information in the catalogue on the disc. It then moves the read head to the correct track and sector on the disc and loads the correct amount of data. When data is saved onto the disc, the computer writes the new catalogue information and then saves the actual data on the disc, starting at the the first free sector. It first checks to see if there is a space between files large enough to hold the new data. If not it will put the data at the end after all the current files.

All the catalogue information such as the filename, its location on the disc, how long it is, etc. is stored on the first two sectors on the first track 00. The catalogue also contains similar information about all the other files on the side as well as the total number of sectors on the side, 400 for a 40 track, 800 for an 80 track disc. All this catalogue information is stored in 512 bytes - and this is the main reason why only 31 files are allowed per side of a disc.

When a file is deleted from the catalogue using the normal DFS commands *DELETE, *WIPE, *DESTROY etc., the computer only deletes the entry in the catalogue. It does not alter or delete any of the actual file on the disc. Beacause it is possible with DISC DOCTOR to load sectors from the disc directly into memory, without looking at the catalogue, it is possible to recover data or a program that has been deleted.

The method of recovering data is explained in detail in the appropriate section under *RECOVER and *RESTORE. However the basics are quite simple. It is necessary to find where on the disc the program started and to have a rough idea of its length. It is useful to remember that any data saved onto the disc always starts at a sector boundary - never part way through a sector. The actual start sector of any file can be found with the *INFO command. If the file has been deleted, then it is possible to use the DISC DOCTOR disc search routines to find the beginning of the file. Once found, the appropriate number of sectors can be loaded into memory, and then the correct section of memory can be saved again onto a fresh disc.

Similarly, it can happen that when the computer crashes and the discs are running (this can occur quite often when testing machine code programs) the computer can totally destroy all the catalogue information. In this case it would be possible to recover all the files one by one and save them onto a fresh disc.

It is important to remember that it is only possible to recover data as described above if the disc has had no other data written over those sectors. If a file is saved onto the disc and then promptly deleted, the data will be recoverable. If then a further file is saved it will overwrite that data. Also the *COMPACT command will prevent deleted files being recovered, as this re-arranges all the data on the disc filling in all the spare sectors. Therefore if something goes wrong or you accidentally delete a file, DO NOT SAVE ANYTHING onto the disc again if you want to be able to recover that data.

DUAL CATALOGUE SYSTEMS

Normally the Acorn DFS only allows 32 files to be recorded on any side of the disc. This is quite limiting especially when using an 80 track system. With a DISC DOCTOR ROM fitted it is possible to have 60 files, almost twice as many, per side of the disc.

Basically the system works by having two catalogues stored on the disc. As mentioned above, a normal disc reserves the first two sectors of the disc for the catalogue information. Using the FORMAT routine built into DISC DOCTOR it is possible to format a special disc that reserves an additional two sectors for an alternative catalogue. The *SWAP command will swap the current catalogue with the alternative one, allowing an additional 30 files to be stored. The additional catalogue is actually stored on sectors 2 and 3 of the first track, right next to the original catalogue. A file is created on these special discs called !.!!!!!!! which serves to protect the additional catalogue.

COMMAND SUMMARY

DIS (<*sta*>) (<*end*>) (<*ofs*>)
Produces disassembly listing of memory on screen or printer.
DISCTAPE <*afsp*> (<*afsp*>)...
Copies all or selected files from disc to tape automatically.
DOWNLOAD <*fsp*> (<*adr*>)
Loads a file into memory and move it automatically to the specified position <*adr*> in memory.
DSEARCH <*str*> <*trk*> (<*trk*><*sct*><*drv*>)
Searches current disc for specified string <*str*>, and display the sector in which it is found.
DZAP (<*trk*>) (<*trk*><*sct*><*drv*>)
An on-screen interactive editing of any disc sector.
EDIT (<*key no.*>)
Display current definitions of any or all function keys, so that they may be edited.
FIND <*str*>
(FOR BASIC PROGRAMS ONLY):-Displays all the line numbers in which the specified string <*str*> occurs.
FORM <*drv*> <*no. trks*> (<*stt*>) (<S>)
Formats specified disc (or any part of it) with the required number of tracks.
JOIN <*fsp*> <*afsp*> (<*afsp*>)...
Joins together all the files specified into a new file named <*fsp*>.
MENU (<*drv*>)
Displays a menu of all files in special directories, and allows user selection of one program for execution.
MOVE (<*dest page*>) (<*src page*>)
Moves a BASIC program in memory from its current position to any specified new position page.
MSEARCH <*str*> (<*adr*>)
Searches memory for the specified string, and displays that area when found.
MZAP (<*adr*>)
An on-screen interactive memory editor.
PARTLOAD <*fsp*> <*ofs*> <*ext*> <*adr*>
Loads a specified part of a file into memory at any position.
RECOVER <*trk*> <*sct*> <*sct*> <*adr*> <*drv*>
Loads a specified sector or sectors of a disc into memory. Used with *RESTORE for recovery of damaged/deleted files.
RESTORE <*trk*> <*sct*> <*sct*> <*adr*> <*drv*>
Transfers a section of memory onto a specified area of a disc. Used with *RECOVER to restore damaged/deleted files.
SHIFT <*src*> <*dest*> <*ext*>
Shifts an area of memory to a different position in memory.
SWAP (<*drv*>)
Swaps catalogues, allowing 60 files per side. Only works with specially formatted discs.
TAPEDISC (<*fsp*>)...
Copies all or selected files from tape to disc,
VERIFY (<*drv*>) (<*no. trks*>) (<*stt*>)
Verifies the disc in the specified drive.

ARGUMENT ABREVIATIONS

<*sta*>	Start address in memory.
<*end*>	End address in memory.
<*ofs*>	Memory offset address.
<*fsp*>	File specification, i.e. a file name in which 'wildcard' characters are NOT permitted.
<*adr*>	An address in memory.
<*str*>	String of characters, which may include control characters in a special form.
<*trk*>	Track number on a disc.
<*sct*>	Either sector number or sector extent i.e. number of sectors on a disc.
<*key no.*>	Function-key number, in the range 0-15.
<*drv*>	Disc drive number, in the range 0-3.
<*no. trks*>	Number of tracks on a disc.
<*stt*>	Start track on a disc.
<*afsp*>	Ambiguous file specification, i.e. a filename in which 'wildcard' characters ARE permitted.
<*dest page*>	Destination page in memory. Only applicable to *MOVE command.
<*src page*>	Source page in memory. Only applicable to *MOVE command.
<*ext*>	Extent/size of memory expressed as the number of bytes.
<*src*>	Source address in memory. Only applicable to *SHIFT command.
<*dest*>	Destination address in memory. Only applicable to *SHIFT command.

GLOSSARY OF TERMS

AMPERSAND	The 'and' sign & located above the six digit on the keyboard, used to show that a numeric argument is in Hexadecimal.
ARGUMENT	A value (numeric or string) passed to a command. Also called a PARAMETER. See section 1
ASCII	American Standard Code for Information Interchange. The method of encoding the character set used in the BBC Micro and most others.
ASTERISK	The 'star' character * located above the colon on the right-hand side of the keyboard, used preceding every DISC DOCTOR command or Operating System command.
BASE	See RADIX.
BINARY	Radix two.
DECIMAL	Radix ten.
HEXADECIMAL	Radix sixteen.
OCTAL	Radix eight.
PARAMETER	See ARGUMENT.
RADIX	(Also called BASE) is the number base, e.g. Decimal, Hexadecimal, Binary, etc. See section 1 for entering numbers with different radices.
ROM	Read Only Memory. The DISC DOCTOR is supplied on a ROM chip which can be erased, more specifically called an EPROM.
SECTOR	A 256 byte section of a disc. There are 10 sectors on every TRACK.
TRACK	10 consecutive sectors recorded on a circular path around a disc. Tracks are split into sectors. There are usually 40 or 80 tracks per side of a disc.