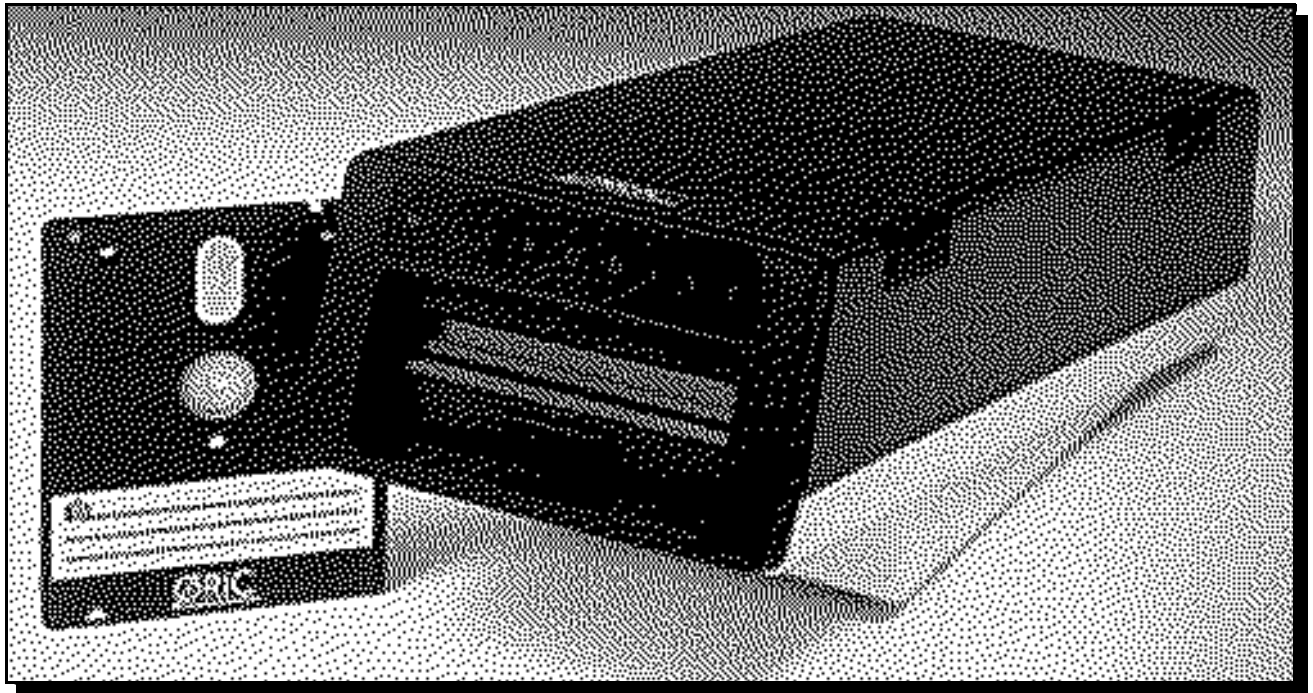


Based on the French SEDORIC V1.0 manual  
by Fabrice Broche and Denis Sebbag

# SEDORIC DOS

Version 3.0

## A USER'S GUIDE



by Jonathan Haworth and Allan Whitaker

Third Edition

April 1996



# SEDORIC DOS

Version 3.0

Copyright 1990, 1991, 1992, 1995, 1996

(1995 and 1996 additions and revisions by Jonathan Haworth)

## C O N T E N T S

INTRODUCTION	1
Conventions and notations	2
Filenames	3
DISC OPERATING COMMANDS	4
Disc directory	5
Loading and saving a file	7
Deleting and modifying a file	9
Initialising and copying of discs	11
Disc configuration	13
EXTENDED BASIC	15
BASIC programming aids	15
Changing the keyboard	19
Function key set-up	21
String commands	27
Error handling routines	29
Text formatting	30
Formatted screen/printing	32
Printer output	33
Graphic instructions	34
User commands	36
Other commands	38
DATA FILES	39
Sequential files	39
Random access files	43
Transfers between buffer and disc	44
Working on the buffer	45
Transfers between fields and variables	46
Disc access	47
Sector reservation	48
INDEX OF DOS KEYWORDS	49
APPENDICES	50
DOS Keyword Addresses and Codes	50
Error Messages	51
Function Key Codes	52
System Variables	53
Disc Structure	54
File Status Byte Coding	55
Keyboard Codes for KEYIF Instruction	55
Switching between RAM and ROM	55
Utility Files on SEDORIC Master Disc	56
Known bugs and limitations - V1.006	58
Developments and solutions - V1.007/V2.0/V2.1	58



## SEDORIC - A USER'S GUIDE

### INTRODUCTION

SEDORIC is a fast, sophisticated Disc Operating System (DOS) for the Oric computer. It is capable of reading or writing 24 kilobytes of code in less than 3 seconds. Despite the sophisticated features of the DOS the amount of memory taken away from the user is minimal. Pages 1 and 2 are not used. On booting the SEDORIC DOS disc the DOS code is loaded into memory addresses #C000 to #FFFF in RAM overlay. The DOS makes use of memory locations in page 0 and page 4 is entirely allocated to SEDORIC. The Oric ROM is switched in and out as needed in order to access BASIC.

Booting SEDORIC carries out a *NEW* command on any program in memory, though a BASIC program can be recovered with the *OLD* command. Before control is passed to BASIC, SEDORIC executes a series of initialisation instructions particular to the disc. These may be modified by the user, to include the loading and execution of your own keyboard file or program to personalise the boot. Two types of discs can be formatted - a 'Master', containing the full DOS, and a 'Slave' which can only be used when the DOS is already in memory. If the DOS is missing, the error message **\*\*\*WARNING\*\* DOS is altered!** is displayed, requiring the insertion of a Master disc into the system drive if the DOS in memory has been affected.

You are strongly advised to make a backup copy of the SEDORIC disc for daily use, preserving the original. To do this type *BACKUP* (or *BACKUP A TO B* if you have two disc drives of the same type) and press RETURN. Press RETURN again and answer 'Y' to the question 'Format target disc Y/N'. Then follow the displayed instructions to carry out the BACKUP.

To display the contents of your SEDORIC disc, type *DIR* and press RETURN. To load a program displayed in the directory, simply type in its name and press RETURN. To exit and reboot SEDORIC, just press the disc drive RESET button.

To save a BASIC program, type *SAVE "NAME"* and RETURN. A *DIR* will prove it has been saved to disc. You will note that the directory of the disc displays the file as NAME.COM. The '.COM' is known as the filename extension and will be covered in more detail in the section on filenames. For now we can ignore it. To recall a program, just type its name (as before), i.e. *NAME* + RETURN. You can also type *LOAD "NAME"* which has the same result.

If you wish to remove a file from the disc, type *DEL "NAME"* and RETURN. To rename a program, use *REN*. Thus *REN "NAME" TO "TEST"* will change the name of 'NAME' to 'TEST'.

To initialise a blank disc for use by SEDORIC only requires that you type *INIT* and RETURN. You will probably come across the message "INSERT MASTER DISC IN DRIVE A". Do so, and press RETURN; the routine you are using will then be loaded into memory. This is necessary because the whole of the DOS is too large for the available memory. Certain commands therefore have their routines stored on disc ready to be loaded when required. Once loaded the command can be used repeatedly until another such command routine has to be loaded (e.g. *BACKUP*) or the computer is switched off. Put in your blank disc, then answer the displayed questions.

As with the BACKUP of a disc, you can make a copy of your programs by using the *COPY* command (e.g. *COPY "TEST"*) and again follow the displayed instructions.

After this brief introduction, you are recommended to read the section entitled 'Disc operating commands' starting on page 4 of the manual to discover the full potential of SEDORIC. Furthermore, SEDORIC provides you with some 60 additional BASIC instructions, and a comprehensive suite of sequential and random access file routines. You will find plenty to keep you occupied while learning to get the best out of SEDORIC!

Note that the SEDORIC disc contains several utilities in addition to the DOS. These are described in detail in the Appendix on p. 54, but take particular note of the CONVERT utility to transfer Oric DOS and Jasmin FT-DOS programs to SEDORIC format.

SEDORIC DOS makes full use of the FUNCT key on the Oric Atmos. In conjunction with the other keys it can provide many functions which are used simply by pressing FUNCT and another key or keys. You can set up your own function keys with the *KEYDEF* and *KEYUSE* commands. Thus, FUNCT and the 7 key resets the screen colours; FUNCT and RETURN together provides automatic numbering of program lines.

A new control character - CTRL-P - has been added to provide an on/off toggle for the cursor flash. Within programs use *PRINT CHR\$(16)*.

Facilities are provided to avoid inopportune system 'crashes'. SEDORIC controls any BREAK and displays a 'BREAK ON BYTE #XXXX' message. The ESC key provides a way out of all interactive commands such as *DIR*, *DEL* and *INIT*.

Whereas all Disc Operating Systems previously offered for the Oric computers have required keywords to be prefixed with the exclamation mark (!) SEDORIC DOS allows the entry of keywords without this need. The exception is following the use of the *QUIT* command which makes the prefixing of DOS keywords or filenames (in direct mode) with the exclamation mark mandatory.

As with BASIC, all DOS keywords must be typed in upper case. BASIC keywords can safely be used in filenames, providing they do not form the start of a filename being loaded in direct mode, e.g. STOPPER. Filenames can also be typed in lower case letters although if BASIC keywords are embedded they must always be typed in upper case, as in 'STOPper'.

### Conventions and notations

The following conventions and notations are used in this manual:

A COMMAND appears in bold or italic capital letters, as does a program line. Key presses are indicated by outline capitals. Optional parameters are enclosed in brackets. A command is entered by pressing the RETURN key.

✓ indicates additional comments for more advanced use of SEDORIC

➡ indicates a bug or other curiosity in SEDORIC

## Abbreviations:

- Drive - the parameter is a drive number - A, B, C or D (as in MS-DOS). The default is the current drive.
- FN - File name required, must not contain wildcards
- FNA - File name ambiguous - File name required, but can include wild card characters (? and \*). It is possible to omit the filename altogether or just to specify the drive name.
- NE - Numeric expression
- AE - Alphanumeric (string) expression
- NV - Numeric variable - a variable name is required
- AV - Alphanumeric (string) variable
- LN - Logical number - refers to a block number used in the control of files.
- FLN - Field name - used in random access file control

The ESC key enables you to exit from all positions where input is required from the user.

## Filenames

A filename is generally a string of characters enclosed in double quotes, or a string expression. An exception is adopted for the loading of files in direct mode, when the filename alone suffices.

A filename may consist of up to three parts:

- (i) the drive to use (optional)  
The drives are designated A to D, and when specified the letter is followed by a hyphen. When not specified the current (default) drive is used. The use of letters to denote each drive allows the drive to be specified in direct mode. The use of numerals would be treated as lines of a BASIC program.
- (ii) the file identification name ('the filename')  
Of up to 9 characters, this must be specified. Only alphabetic and numeric characters may be used, save for the wildcard characters '?' and '\*'.
- (iii) the file extension (optional)  
When specified, this is separated from the filename by a full stop. It comprises up to three characters, enabling files to be categorised (e.g. .BAS, .BIN etc). When no extension is specified the default is assumed (see the *EXT* command). On initialisation the default is .COM.

Ambiguous filenames (FNA) are valid for such commands as *DIR*, *DEL* and *COPY* etc. The '?' is substituted for individual letters, the '\*' for a group of characters to the end of the name. So *DEL "\*.BAS"* deletes all files with the extension .BAS, whereas *DIR "M\*"* lists all filenames starting with the letter M sharing the default extension. The asterisk does not extend beyond the part of the filename it is placed in. In DOS commands where ambiguous filenames are allowed it is possible to omit the drive, extension or filename characters, or to substitute wildcard characters (e.g. \*.\*). In the latter example, all files on the default drive are accessed.

## DISC OPERATING COMMANDS

**(drive)-**

On its own, it sets the default (or current) drive. Used with a filename it allows temporary access to the specified drive but the default drive is unchanged. The drive name is optional in the filename. If used with the *LOAD* command it must be within the quotes for the filename. If not specified, the default drive is accessed.

*B-* selects drive B as the default  
*A-MENU* load MENU(.COM) from drive A, but B would remain the default drive  
*LOAD "C-HELP.BAS"*

**EXT AE**

Specifies the default file name extension to 3 characters. On initialisation the default name is .COM

*EXT "BAS"*

**EXT ?**

Displays the current default extension.

➡ Curiously, this can be entered as *EXT PRINT* as well!

**EXT**

Resets the default extension to .COM



## DISC DIRECTORY

### **DIRFNA**

Catalogues the disc. The display can be interrupted by pressing a key; press the SPACE BAR to restart, ESC to exit. Each filename is shown with the number of sectors it occupies. Protected files are marked with a 'P'. The number of files is limited only by disc capacity.

```
DIR  
DIR "*.BAK"  
DIR C
```

### **CHKSUM NFA (,AUTO)**

An extended *DIR* command that lists the filenames on the disc in the default or specified drive together with their start, end and execution addresses, their file type and a checksum of all the bytes in the file. The latter is useful to see whether two files are identical or not. See *LOAD, V* on page 7 for an explanation of file status information.

If the *,AUTO* parameter is omitted, a key must be pressed for the next file to display. If it is included, all relevant files will display automatically, but may be halted by pressing SPACE and restarted by pressing any other key. ESC exits at any time.

```
CHKSUM "PROG.BAS"  
CHKSUM "*.TXT", AUTO
```

### **VIEWHIRES NFA (,AUTO)**

View all HIRES screens on the disc in the default or specified drive. The command benefits from the use of the convention that all HIRES screens are saved with the *.HRS* extension.

If the *,AUTO* parameter is omitted, a key must be pressed for the next file to display. If it is included, all relevant files will display automatically, but may be interrupted by pressing SPACE and restarted by pressing any other key. ESC exits at any time.

```
VIEWHIRES "SCREEN.HRS"  
VIEWHIRES "*.HRS", AUTO
```

**LDIR FNA**

Catalogues the disc sending the output to the printer.

**SEARCH FNA**

Checks if a file is on disc. If so, the system variable EF=1, if not 0. If EF is to be used in a logical expression then -EF must be used to achieve the correct logic values (-1=TRUE, 0=FALSE). Ambiguous filenames may be used, when EF=1 if at least one match is found.

```
10 SEARCH "TEST"  
20 IF EF=1 THEN LOAD "TEST" ELSE PRINT "File not found"
```

## LOADING AND SAVING A FILE

**LOAD FN (, A, NE) (, V) (, J) (, N)**

Loads a file. Options:

, **A**, NE loads the file at the address specified in NE  
, **J** joins the file to a BASIC program  
, **N** stops autorun after loading  
, **V** shows the file status, but does not load it (see also *CHKSUM*).

File status is shown in the format *SSSS FFF SS EEEE*

where *SSSS* is start address of file (no. of records for random access file)

*FFFF* is end address of file (length of random access file)

*SS* is the file status byte -

80 = BASIC

81 = BASIC auto

40 = Machine code

41 = Machine code auto

(see the Appendix on p. 55 on file status byte coding)

*EEEE* is execute address for a machine code program - default value is 0.

Alternatively the file status can be obtained when loading a file by examining the system variables ST (STart address), ED (EnD address), FT (File Type), and EX (EXecution address). Note that these variables will be cleared if a BASIC program is allowed to run on loading. They may also be corrupted if a file is loaded into the area of memory occupied by the variables. To preserve them, use the ,V option which will only load in the status byte and not the whole file.

*LOAD "TEST"*

*LOAD "B-ROUTINES", J, V*

*LOAD A\$ + ".TXT", A#4000*



The options ,A and ,J are mutually exclusive, as are ,V and ,N. The option ,V used with ,A or ,J displays the new addresses of the start and end of file - very practical.

**(FILENAME) (, A, NE) (, N) (, J) (, V)**

It is unnecessary to use *LOAD* unless the filename is expressed as a variable, e.g. *LOAD A\$*. It is enough simply to type the filename to load it - provided the filename does not start with a BASIC keyword or a number. If it does, specify the drive, e.g. *A-LOADER*. Note that after *QUIT* is used, the ! must be used as a prefix to the filename.

*TEST*

*B-DESIGN.SCR, A#1000*

*ROUTINE.BAK, V*

```

SAVE FN (, A NE) (, E NE) (,T NE)
(,AUTO)
SAVEO " " " " " "
SAVEM " " " " " "
SAVEU " " " " " "

```

Saves the file with the name FN to disc.

The four save commands only have different effects if the filename already exists on disc. The syntax is as for *CSAVE*; if no option is used, the current BASIC file will be saved. The ,A and ,E extensions are for machine code file start and end addresses only. As on cassettes, it is the status of the saved file which determines whether it autoruns or not. The status can easily be altered subsequently (see the *STATUS* command). Using , T NE enables an execution address to be specified that is not the file start address. This is not operative for a BASIC program.

If the file already exists:

- SAVE* gives a 'File already exists' message
- SAVEO* overwrites the existing file unless it is protected
- SAVEM* the new file saved is appended to the existing file on disc
- SAVEU* the existing file is preserved as a .BAK file. Any existing .BAK file is lost.

```

SAVE "TEST", AUTO
SAVEM "DESIGN.HRS", A#9800, E#BFDF

```

✓ The *SAVEM* command does no more than juxtapose the new file with the old one under one name. It has nothing to do with the ,J option for *LOAD*, for example. A *LOAD* will load both sections of the new file together.

```

ESAVE FN

```

Saves the current screen (TEXT or HIRES). The computer must be in the same mode when the file is reloaded.

```

ESAVE "STORY.TXT"
ESAVE "DESIGN.HRS"

```

✓ This command is equivalent to *SAVEU FN, A#BB80, E#BFDF* in Text mode, and *SAVEU FN, A#A000, E#BF3F* in Hires mode.

## DELETING AND MODIFYING A FILE

### **DEL FNA**

Deletes an unprotected file. Wild cards are permitted, in which case confirmation is requested for each file. Use ESC to exit.

```
DEL "TEST.COM"  
DEL "*.BAS"  
DEL  
DEL B
```

### **DESTROY FNA**

DANGEROUS! As for *DEL*, but with no request for confirmation. Protected files are not affected. *DESTROY* alone will delete all files on disc. If used in error, RESET to salvage as many files as possible.

```
DESTROY "*.BAK"
```

### **DELBAK (drive)**

Deletes all .BAK files without asking for confirmation.

```
DELBAK  
DELBAK B
```

### **REN FNA (old) TO FNA (new)**

Renames a file. Both the new and old filenames are displayed. Wildcards are allowed, provided they are in the correct corresponding positions.

```
REN "PROG.DAT" TO "GAME.COM"  
REN "*.COM" TO "*.BAS"  
REN "PROG?" TO "TEST?"
```

## STATUS FN (, A NE) (, T NE) (, AUTO)

Allows you to change the status of a file without having to load and save it, following the syntax for *SAVE*. If no option is specified, autorun is removed.

- , A NE alters the loading start address; the DOS calculates the new end address itself
- , T NE forces a machine code file to AUTOrun from the address specified in NE. For BASIC files is equivalent to ,AUTO
- , AUTO forces a BASIC or machine code file to AUTOrun from the start.

```
STATUS "TEST"  
STATUS "MENU", AUTO  
STATUS "B-PROGMC", A#4000, AUTO  
STATUS "PROG.COM", T#5000
```

✓ If files have been grouped using *SAVEM* or *COPYM*, this command only affects the first file, which is also then the only one initiated on a *LOAD*.

## PROT FNA

Protects a file from deletion or overwriting. A letter 'P' is displayed by the filename in the directory.

```
PROT "TEST"
```

## UNPROT FNA

Removes protection from the file and the 'P' from the directory.

```
UNPROT "*.COM"  
UNPROT B
```

## INITIALISING AND COPYING OF DISCS

**INIT (drive , no. of sectors , no. of tracks , S or , D)**

To initialise a disc, follow the on-screen instructions (use ESC to abort). The number of sectors per track and the number of tracks may be specified along with , *S* or , *D* to indicate a single or double-sided drive. The number of tracks per side must be between 21 and 101 inclusive, and the number of sectors per track between 16 and 19 inclusive. Any value outside these parameters will produce an error message. The maximum number of tracks on a 3½" or 5¼" disc is recommended to be 82. Specifying 16 or 17 sectors per track gives the highest reliability and fastest speed. The default value is 17 sectors, single-sided, but this can be preset to a new value, associated with each disc, using the command *DTRACK*.

You are offered the options to format and name the new disc, to enter an initial command statement to run on booting, and to make it a Master disc (with full DOS) or a Slave disc (saving 90 sectors).

*INIT INIT B,17,42 INIT C,17,80,D*

Several replies are required from the user:

- Format (Y/N): Answer 'Y' to format or 'N' to go directly to the next stage
- Name: Enter a name for the disc. To include attributes in the disc name use CTRL-Z followed by the appropriate code as used with the ESC key in direct mode or CHR\$(27) in program mode.
- Init Statement: Enter the instruction(s) to execute on booting the disc. If none, the disc will go straight to BASIC command level
- Master Disc (Y/N) Enter 'Y' for a Master disc, 'N' for a slave disc. A Master disc has a full copy of the DOS, a slave disc must only be used with the DOS loaded, but provides an extra 90 sectors.

✓ Although the parameters are not obligatory, they cannot be used beyond an omitted parameter. Thus *INIT - INIT A - INIT, 16 - INIT 18,41 - INIT 19, 40, D* are acceptable, but *INIT A, , 42, S - INIT A, , ,S* are not.

✓ Formatting a disc will erase any existing files on disc. *INIT* affects memory from #3000 to #BoFF, so save any program in memory before using it.

**WARNING!** If asked to insert a Master disc be sure to replace it with the disc being initialised before answering the *Format (Y/N)* request, otherwise you could reformat your master disc if it is not write-protected.

**BACKUP (drive) (TO) (drive)**

Copies one disc (the source) to another (the target). It affects memory from #0600 to #B4FF. Use identically formatted discs or a 'DISC I/O' error is generated. On a single drive you must exchange the source and target discs as prompted. It is advisable to write-protect the source disc.

*BACKUP*  
*BACKUP TO B*  
*BACKUP C*

**COPY (COPYM or COPYO) (FNA) (TO FNA) (, N) (, C)**

Copy files - *M* and *O* work as for *SAVE*.

*COPY* checks the file is not already on the target disc. If so, a 'FILE ALREADY EXISTS' message is displayed

*COPYO* overwrites such an existing file, otherwise works as *COPY*

*COPYM* permits a merge copy to one file of several files by the use of wildcards - if the target file already exists, the source files are merged to it. If not, a new file comprising all the source files is created.

*COPYM "A-\*.CDE" TO "B-PROG"*

, *N* copies files on the source disc itself, omitting the change disc prompts.

, *C* asks for Y/N confirmation for each file to be copied

With wildcards, two uses exist:

With *COPY* and *COPYO* the source filename can contain wildcards. If so, there must be a corresponding wildcard in the target filename. The various files so specified are then copied with their correct source filenames. The only exception to this is that you may put a wildcard in the source filename and omit a target filename, or compose the target filename entirely of wildcards (e.g. \*.\*).

With *COPYM* the source filename can contain wildcards, but never the single target filename.

*COPY "A-TEST" TO "B-PROG"* copies the file TEST from drive A to drive B and renames it PROG.

*COPY "B-\*.COM" TO C-\*.CDE"* copies all files with the .COM extension on drive B to drive C, giving them the new extension .CDE.

✓ Note that while *CLOAD*, *J LOAD*, *J* and *MERGE* work only with BASIC files, *COPYM* works with all file types. *COPY* without a source filename copies all files from the current disc. The target filename may also be omitted (in which case the files are copied to the current drive), except of course with *COPYM*, which must have a specific target filename.



## DISC CONFIGURATION

When first receiving your SEDORIC V2.0 Master disc you need to check the configuration of your disc system and correct it if it is set wrongly on the master disc. You can check the configuration by using the *DSYS* command. For instance, if you have two 3" disc drives connected and when you checked, using *DSYS*, your system was set to the following:-

```
drive A      ...    42 tracks 17 sectors, single-sided
drive B      ...    82 tracks 17 sectors, double sided
```

then use the *DTRACK* command to set the correct configuration on the master disc (ensure the master disc is not write-protected). This is achieved by entering:-

*DTRACK 42, 42* or *DTRACK A, 42, 42*

To configure one double-sided 3½" disc drive it would be:-

*DTRACK 80; D*

**SYS**

Displays the DOS configuration in memory (not that on disc) and drives connected.

**DSYS (drive)**

Displays the DOS configuration saved on the specified disc.

*DSYS B*

**DNAME (drive)**

Modifies the name of the disc in the specified drive (or default if none is specified), displaying the old name and then requesting and writing the new one. The maximum is 21 characters. To add attributes, use CTRL-Z as for the *INIT* command (see page 11).

*DNAME*  
*DNAME A*

**INIST (drive)**

Modifies the initialisation instructions on the specified disc, allowing you to personalise the start-up routines. Displays the old instructions (if any) and requests the new one. Maximum length is 60 characters, and attributes may be entered. Note the bug dealt with under *PR*, page 33.

*INIST A*  
*INIST B*

**DKEY** (drive) (, A) (, S)

Sets the disc start-up keyboard configuration - QWERTY and accents off if no option is specified, AZERTY with , A , with accents with , S . The comma is still necessary when no drive is specified.

*DKEY B, A*  
*DKEY, S*

**TRACK** (NE<sub>A</sub>) (, NE<sub>B</sub>) (, NE<sub>C</sub>) (, NE<sub>D</sub>)

Modifies (only in memory) the disc drive configuration by setting the number of tracks for each drive. A value of "0" disables the drive, which then on access will give a 'DRIVE NOT IN LINE' error. To configure a drive as double-sided, add ";D" to the number of tracks. The default is single-sided. If skipping a drive the comma must still be inserted.

*TRACK 41,,42; D* sets drive A to 41 tracks and drive C to 42 tracks double-sided

**DTRACK** (drive) (NE<sub>A</sub>) (, NE<sub>B</sub>) (, NE<sub>C</sub>) (, NE<sub>D</sub>)

As the above command, but the change is made on disc and not the DOS in RAM. The format is then loaded on each initialisation.

*DTRACK A, 42*

**DNUM** (drive) (, NE) (, NE)

Modifies on the disc in the specified drive the default initialisation values for respectively the first line and step values for the renumbering commands *NUM* and *RENUM* and for auto line numbering. If the drive letter is omitted the first comma must still be used, as must the second if you only wish to alter the step value.

*DNUM B, 100, 10*  
*DNUM , , 10*

✓ *DNUM* alone has the effect only of loading Bank 5 from disc into memory. This could enable the use of the Bank 5 commands within a machine code program.

**SYSTEM** (drive)

Specifies the disc drive on which is the DOS containing transient command files. The default drive on booting is always A.

*SYSTEM D*

## EXTENDED BASIC

### BASIC PROGRAMMING AIDS

**RENUM** (NE<sub>1</sub>) (, NE<sub>2</sub>) (, NE<sub>3</sub>) (, NE<sub>4</sub>)

Renumbers all or a block of a BASIC program. The parameters should be entered in the following order:

- NE<sub>1</sub> new first line number
- NE<sub>2</sub> line increment required
- NE<sub>3</sub> and NE<sub>4</sub> the first and last line numbers of the program to be renumbered.

If any values are not declared the default values are:

Start line number	=	100
Increment	=	10
First/last line	=	the first/last lines of the existing program.

The values for the origin and step can be modified with the *DNUM* command. All GOTO, GOSUB, ON GOTO, ON GOSUB, THEN, ELSE, RUN, and RESTORE commands are correctly modified. Numeric expressions and variables (e.g. *GOTO 100\*A: GOTO AB*) are NOT modified. Use ON GOTO or ON GOSUB instead.

<i>RENUM</i>	=	<i>RENUM 100, 10</i>
<i>RENUM1000</i>	=	<i>RENUM 1000, 10</i>
<i>RENUM,,,10000</i>	=	<i>RENUM 100, 10, 0, 10000</i>
<i>RENUM1000,,2300</i>	=	<i>RENUM 1000, 10, 2300</i>

*RENUM100,10,0,10000* is the full command.

✓ This command affects only the actual line number. It does not reposition the block of lines in memory, e.g. *RENUM 100,10,20,30* gives:

10 REM EXAMPLE	10 REM EXAMPLE
20 PRINT "TEST"	100 PRINT "TEST"
30 PRINT "EXAMPLE"	110 PRINT "EXAMPLE"
40 END	40 END

To move lines, cut out the lines to move (in continuous blocks), renumber them, save them, and then reunite them with the original using *LOAD, J* or *MERGE*.

➡ In direct mode it is possible to enter line numbers up to 63999 (#F9FF). By using *RENUM* lines can be numbered up to 65534 (#FFFE). But beware, because certain commands will give an 'ILLEGAL DIRECT ERROR' if they are found in a line with a number above 65729 (#FEFF). Numbers #FF00 and above give such an error because the #FF acts as the direct mode indicator.

➡ There is no validation of the parameters for *RENUM*. Thus a step of 0 works, but all lines have the same number! Save your program before each *RENUM* to be safe.

➡ The BASIC token RESTORE (#9A) calls the ROM, and is taken account of by *RENUM* but not correctly dealt with because of the lack of an argument! Always use *"!RESTORE"*.

**MERGE FN ( , L)**

Merges the BASIC program in memory and the BASIC program specified. If the resultant program is too large for the available memory an 'OUT OF MEMORY' error will be generated. All variables are preserved, but as usual any functions defined by DEF FN are lost. If the same line number appears in both programs, the one in memory will be used and a 'LINE ALREADY EXISTS' message displayed. The command can be speeded up by omitting the on-screen information if you add , *L*.

If using *MERGE* within a program, ensure no line will be inserted before the line containing the *MERGE* command, otherwise a 'SYNTAX ERROR' is generated.

*MERGE "PROG"*  
*MERGE "TEST", L*

**SEEK (AE) ( , S) ( , M)**

*SEEK AE* lists all program lines in which the string AE exists. The listing can be interrupted with SPACE or CTRL-C as usual. The string must not include the ASCII nul code CHR\$(0), otherwise an 'INVALID STRING ERROR' is generated.

To search for BASIC keywords AE must contain the token value (see *TKEN*) rather than the text equivalent (e.g. #80 for *END*). The string may include wildcards; the wildcard character is '£' because of the frequent use of '?' in BASIC.

*SEEK CHR\$(#80)*

The maximum string length is 79 bytes.

Option , *S* omits the screen listing, giving simply a total number of occurrences. Option , *S*, *M* omits any screen information. In each case the system variable SK contains the number of occurrences.

*SEEK* alone (used after a *SEEK AE*) displays each line occurrence one at a time for each entry of the command. The prescribed string remains in memory until another *SEEK AE* or a RESET.

Example: Given the following program in memory

```
10  REM TEST SEEK SEDORIC 0.0
20  PRINT "HELLO SEDORIC 1.0!"
30  END
```

*SEEK "HELLO"* lists line 20

*SEEK "HELLO",S* displays 1 Found

*SEEK "ORIC £.0"* lists lines 10 and 20

then *SEEK* lists line 10, and another *SEEK* lists line 20

*SEEK CHR\$(#BA)* lists line 20 since #BA is the token for the PRINT command

✓ Useful tokens are: 191 for CALL, 192 for !, 182 and 183 for CLOAD and CSAVE, 230 and 231 for PEEK and DEEK, 185 and 138 for POKE and DOKE, and 157 and 39 for REM and '.

## CHANGE AE<sub>1</sub> TO AE<sub>2</sub>

Replaces all occurrences of string AE<sub>1</sub> by the string AE<sub>2</sub> in a BASIC program. As for *SEEK*, the strings must not contain the ASCII Nul code CHR\$(0), but may contain BASIC tokens. The maximum length of string is 78 bytes. The "£" is the wildcard character. If the '£' is present in the same place in both strings, that character will not be modified; if it is only present in the first string, it will be replaced by the character in the same place in the second string.

Note that use of the wildcard in the target string will be treated literally. Use of the wildcard enables several strings to be replaced by one, but not a 'selective' change as with the *REN* command for filenames.

If the BASIC program is made too large for the available memory, the 'OUT OF MEMORY ERROR' is generated and execution halted. If the string exchange would result in a line being too long then the error message 'LINE; XXXX ?STRING TOO LONG ERROR' is displayed.

*CHANGE CHR\$(#BA) TO CHR\$(#8F)* changes all PRINT commands to LPRINT.  
*CHANGE CHR\$(157) TO CHR\$(39)* changes all REMs to '  
*CHANGE "ORIC" TO "ORIC ATMOS"*

In the example program under *SEEK*, the command:

*CHANGE "ORIC £.o" TO "BOOM !"* gives the lines:

```
10    REM TEST SEEK SEDBOOM !
20    PRINT "HELLO SEDBOOM !"
```

✓ The second string AE<sub>2</sub> may be empty, but not the first AE<sub>1</sub>. This can be used to remove all occurrences of a command, e.g. to remove all REMs use

*CHANGE CHR\$(157) TO ""*

✓ The second string AE<sub>2</sub> need not be tokenised for BASIC keywords. Thus to alter the command WAIT 50 to WAIT 100 at every occurrence use

*CHANGE CHR\$(181)+" 50" TO "WAIT 100"*

✓ Remember that *TOKEN* is useable only in program mode.

**DELETE (NE<sub>1</sub>) - (NE<sub>2</sub>)**

Deletes a block of lines from a program without losing variable values. The syntax is as for LIST, but the "-" is mandatory. Without it *DELETE* causes a SYNTAX ERROR. *DELETE* is equivalent to a NEW. If using *DELETE* within a program, ensure the command is not placed in or after the block to delete. User functions defined by DEF FN located in the deleted block will be lost, but variable values are preserved.

*DELETE 100-100* deletes line 100  
*DELETE -1000* deletes all lines up to line 1000  
*DELETE 234-987* deletes lines 234 to 987

➡ If a parameter exceeds 63999 (possibly after a *RENUM*), an ILLEGAL QUANTITY ERROR follows. It is then impossible to delete certain lines.

**NUM (NE<sub>1</sub>) (, NE<sub>2</sub>)**

Sets the parameters for automatic line numbering. NE<sub>1</sub> is the start line, and NE<sub>2</sub> the increment. Each press of FUNCT + RETURN ends the current line and numbers the next.

The default values are as for *RENUM*, start line 100, increment 10. They can be modified by *DNUM*.

*NUM 1000* numbers in tens from line 1000  
*NUM, 5* numbers by fives from line 100  
*NUM 3300, 20* numbers by twenties from line 3300

✓ *NUM* alone is very useful - it restores the default values as the current ones.

➡ Be aware that there is no validation of parameters. Thus any line number, even above 63999 is accepted - beware!

**NUM END**

If this command is entered, the DOS automatically finds the number of the last line of a BASIC program already in memory, and starts automatic line numbering at the line following (line number plus default increment).

## CHANGING THE KEYBOARD

### **KEY SET or KEY OFF**

Inhibits (*OFF*) or authorises (*SET*) the keyboard. Switching the keyboard off speeds up a program by 20%.

✓ Beware of using *KEYOFF* in direct mode - the keyboard will no longer respond. Do not use a *WAIT* command after *KEYOFF*, since *WAIT* uses the system clock which is no longer updated after *KEYOFF*. Use a *FOR...NEXT* loop instead.

### **KEYIF NE GOTO or KEYIF NE THEN**

Seeks a keypress set by the argument *NE*. Works even if the keyboard is inhibited or several keys are pressed together. The syntax is as for an *IF THEN ELSE*.

*NE* is a special code used to speed execution, listed in the Appendix on page 55.

```
100 KEYIF #84 THEN SHOOT
...
200 KEYIF AB THEN 100 ELSE PING
```

### **QWERTY**

Sets the keyboard to normal (usually after an *AZERTY*) and executes an *ACCENT OFF*. Inoperable in Hires mode.

### **AZERTY**

Sets to French keyboard -

Q	becomes	A
W		Z
A		Q
Z		W
M		;
;		M

It also effects an *ACCENT SET*. *KEY\$* is not affected - it reacts to the *QWERTY* keyboard even after an *AZERTY* command. *CTRL* key combinations follow the *AZERTY* key setup, but function key commands remain in *QWERTY* setup.

✓ This command does not work on the Oric-1 nor can it be used in Hires mode.

**ACCENT SET / ACCENT  
OFF**

Redefines certain characters to show accents on screen. They have been chosen to reflect the ASCII characters used by most printers for the French character set accents.

ASCII	NORMAL	ACCENTED
40	@	à
5C	\	ç
7B	{	é
7C		ù
7D	}	è
7E	■	ê

The command is not usable in Hires mode, but the character set will be preserved when selecting HIRES from TEXT mode. *ACCENT OFF* resets *ACCENT SET*.

✓ The 'ê' is not available directly from one key. By default it can be obtained by using FUNCT - 9, but this may be displaced if you create your own keyboard file (see next page).



## FUNCTION KEY SET-UP

SEDORIC ships with a comprehensive function key set-up installed. You can, however, design your own keyboard set-up by assigning specific effects to each key or key combination.

A function can be assigned to any key in two ways. The first is by the allocation of a pre-determined command using *KEYDEF*, the second the allocation of a command defined by the user utilising *KEYDEF* and *KEYUSE*. To use these commands effectively it is necessary to understand how they interrelate. When a key, FUNCT + key, or FUNCT + SHIFT + key are pressed, a unique keyboard code is generated, as indicated in the following table. Note that there are gaps in the table, indicated by a lower case 'm'. Incidentally, the left hand column shows the codes used by the KEYIF instruction.

### Keyboard codes

Codes for		Index to KEYDEF table for	
		key + FUNCT	key + FUNCT + SHIFT
key alone	0 1 2 3 4 5 6 7 8 9 A B C D E F		
#80 to #8F	7 J M K ■ U Y 8 N T 6 9 , I H L	#C800 to #C80F	#C840 to #C84F
#90 to #9F	5 R B ; . O G o V F 4 - 1 P E /	#C810 to #C81F	#C850 to #C85F
#A0 to #AF	m m c m g f m s 1 e Z m - d A r	#C820 to #C82F	#C860 to #C86F
#B0 to #BF	X Q 2 \   ] S m 3 D C ' - [ W =	#C830 to #C83F	#C870 to #C87F
<p><b>Legend:</b> c CTRL, d DEL, e ESC, f FUNCT, g left SHIFT, m code missing, r RETURN, s right SHIFT, and ■ SPACE. Example: #80 corresponds to the "7" key and #BF to the "=" key. Certain codes (example #A0) do not represent any key.</p>			

By default SEDORIC assigns function codes (spanning #00 to #FF in the body of the KEYDEF table below) to each of the key presses in the two right hand columns in the table above (and hence to the key combinations themselves). The KEYDEF table spans values from #00 to #3F for FUNCT + key combinations and from #40 to #7F for FUNCT + SHIFT + key combinations. To work out the effect of a combination, firstly find the key combination on the right of the tables below, then read off the corresponding function code on the left of the tables (the first column is the address of the table in SEDORIC ROM, #C800 to #C87F). The many '00' entries in the KEYDEF table correspond to a space character.

### KEYDEF Table

Function codes    FUNCT + key:

C800- 07 45 57 4B 00 18 07 08 7 J M K ■ U Y 8  
 C808- 59 7B 06 09 00 42 41 52 N T 6 9 , I H L  
 C810- 05 66 25 00 00 5B 27 00 5 R B ; . O G o  
 C818- 1B 3F 04 0A 00 5E 3D 0D V F 4 - 1 P E /  
 C820- 00 00 00 00 00 00 00 00 m m c m g f m s  
 C828- 01 00 08 00 00 00 22 FF 1 e Z m - d A r  
 C830- 6D 62 02 0C 00 0F 72 00 X Q 2 \ | ] S m    FUNCT + ] = DOKE#  
 C838- 03 31 29 00 00 0E 1E 0B 3 D C ' - [ W =



**2<sup>nd</sup> series: 16 predefined commands (codes #10 to #1F)**

<u>Function</u>	<u>Code</u>	
C980-10	48 45 58 24 <b>A8</b>	HEX\$(
C985-11	43 41 4C 4C <b>A3</b>	CALL#
C98A-12	54 45 58 54 <b>8D</b>	TEXT <b>CR</b>
C98F-13	46 4F 52 49 3D 31 54 <b>CF</b>	FORI=1 <b>TO</b>
C997-14	4C 45 46 54 24 <b>A8</b>	LEFT\$(
C99D-15	4D 49 44 24 <b>A8</b>	MID\$(
C9A2-16	52 49 47 48 54 24 <b>A8</b>	RIGHT\$(
C9A9-17	53 54 52 24 <b>A8</b>	STR\$(
C9AE-18	55 4E 50 52 4F 54 <b>8D</b>	UNPROT <b>CR</b>
C9B5-19	<b>Eo</b>	©
C9B6-1A	55 53 49 4E <b>C7</b>	USING
C9BB-1B	56 49 45 57 48 49 52 45 53 <b>A2</b>	VIEWHIRES
C9C5-1C	56 55 53 45 52 <b>8D</b>	VUSER <b>CR</b>
C9CB-1D	57 49 44 54 <b>C8</b>	WIDTH
C9D0-1E	57 49 4E 44 4F <b>D7</b>	WINDOW
C9D6-1F	21 52 45 53 54 4F 52 <b>C5</b>	!RESTORE

**3<sup>rd</sup> series: SEDORIC DOS keywords (codes #20 to #7F) - See Appendix on p.48**

Code#		Code#		Code#		Code#	
21	APPEND	39	ERRGOTO	54	LDIR	6C	SWAP
22	AZERTY	3B	ERROR	55	LTYPE	6D	SEEK
23	ACCENT	3C	ERR	56	LCUR	6F	STRUN
24	BOX	3D	ESAVE	57	MOVE	70	SYSTEM
25	BACKUP	3E	EXT	58	MERGE	71	STATUS
26	BUILD	3F	FIELD	59	NUM	72	SAVEU
27	CHANGE	40	FRSEC	5A	OUT	73	SAVEM
28	CLOSE	41	HCUR	5B	OLD	74	SAVEO
29	COPY	42	INIT	5C	OPEN	75	SAVE
2A	CREATEW	43	INSTR	5D	PUT	76	SEARCH
2B	CRESEC	44	INIST	5E	PROT	77	SYS
2D	DELETE	45	JUMP	5F	PR	78	SMAP
2E	DESTROY	47	KEYIF	60	PMAP	79	TKEN
2F	DELBK	48	KEYUSE	61	QUIT	7A	TAKE
30	DEL	49	KEYDEF	62	QWERTY	7B	TYPE
31	DIR	4B	KEYSAVE	63	RESUME	7C	TRACK
33	DNUM	4C	KEY	64	RESET	7D	USER
34	DNAME	4D	LINE	65	REWIND	7E	UNTKEN
35	DKEY	4E	LSET	66	RENUM	7F	USING
36	DSYS	50	LUSING	67	REN	(81)	VUSER
37	DTRACK	52	LINPUT	69	RANDOM	(82)	WIDTH
		53	LOAD	6A	RESTORE	(83)	WINDOW

**4<sup>th</sup> series: BASIC keywords (codes #80 to #FD) - See Atmos manual p.288**

Code#		Code#		Code#		Code#	
80	END	9E	HIMEM	BC	LIST	DA	FRE
81	EDIT	9F	GRAB	BD	CLEAR	DB	POS
82	STORE	A0	RELEASE	BE	GET	DC	HEX\$
83	RECALL	A1	TEXT	BF	CALL	DD	&
84	TRON	A2	HIRES	Co	!	DE	SQR
85	TROFF	A3	SHOOT	C1	NEW	DF	RND
86	POP	A4	EXPLODE	C2	TAB(	Eo	LN
87	PLOT	A5	ZAP	C3	TO	E1	EXP
88	PULL	A6	PING	C4	FN	E2	COS



You can of course amend and save to disc the current keyboard using KEYDEF, KEYUSE and KEYSAVE; you can also amend the .KEY files directly on disc using NIBBLE. You can in this way create entirely new keyboard files to suit your own needs.

### KEYDEF NE

Permits the definition of function keys. The argument is the function code, assigning to the key(s) pressed the effect listed in the above translation tables (or in the case of codes 00 to 15 substituted with KEYUSE. After entering the command, press the key to which you wish to assign the function. If you press just the key, the function is accessible by pressing FUNCT and the key. If you press SHIFT and a key, the function is accessible by pressing FUNCT + SHIFT and the key.

To recap, the function codes are:

- Codes 0 to 15 for user-definable commands (with *KEYUSE*)
- Codes 16 to 31 for pre-defined commands (see above)
- Codes 32 to 127 for the DOS keywords (see above and the Appendices)
- Codes 128 to 246 for the BASIC keywords (see above and in the Oric manual)
- Code 254 for FUNCT+DEL (delete keyboard buffer)
- Code 255 for FUNCT + RETURN (auto line numbering)

*KEYDEF 0* assigns to the key you then press the command defined by *KEYUSE 0,"xxxx"*

*KEYDEF #80* assigns to the key pressed the BASIC keyword END (#80 is the hex. code of END). You could equally use *KEYDEF 128*.

### KEYUSE NE, AE

Only works on the Atmos. It permits the definition of the 16 user commands (with one or more commands each).

NE is the command code (0 to 15), AE is the command string (the maximum number of characters is 16).

The string can include all ASCII characters except 0 (null) and characters above 127. It can include CHR\$(13) which simulates a RETURN.

*KEYUSE 0, "PAPER 0:INK 7" + CHR\$(16) + CHR\$(13)*

An example of setting FUNCT and the 1 key to list all the filenames with the extension ".COM" is set out in the three steps that follow.

- 1 Enter *KEYUSE 0, "DIR" + CHR\$(34) + "\*.COM" + CHR\$(34) + CHR\$(13)*
- 2 Enter *KEYDEF 0*
- 3 Press key '1'

Now if you press the FUNCT key with the '1' key, the files will be listed to the screen.

## KEYSAVE FN

Saves the current function key configuration on disc. Equivalent to *SAVE FN, A#C800, E#C9DD*. To recall a saved configuration, simply load it.

✓ *KEYSAVE* automatically behaves like *SAVEU* in that a backup file is created if you save a file to a disc having a file of the same name on it.

*KEYSAVE "TEST.KEY"*

## VUSER

Displays the defined command strings for the 16 user commands. Control characters appear in inverse video - e.g. M inverse for RETURN because CTRL-M is equal to RETURN. A space is inserted before control codes.

✓ This does not show to which keys the user commands are allocated. This would be suitable for an upgrade if it is required by users.

## STRING COMMANDS

**INSTR** AE<sub>1</sub> , AE<sub>2</sub> , NE

Searches for the first occurrence of the string AE<sub>2</sub> within the string AE<sub>1</sub> , starting at the character position specified in NE. The position is returned in the system variable IN.

If the string to be examined is empty or if the string sought is "" or is not present, IN contains 0. If the parameter entered in NE is 0 or greater than the length of the string to be examined an "ILLEGAL QUANTITY ERROR" is returned. String variables may be used.

```
10 INSTR "NICE DAY", "DAY", 1  
20 PRINT IN
```

gives the result 6.

**TKEN** AV

Encodes (in program mode only) the string variable as a suite of BASIC and/or SEDORIC instructions which can be executed by the use of *STRUN*. The variable must contain the string to encode; after execution it contains the encoded string.

The string to encode can contain up to 78 characters, the resultant string up to 67 bytes (BASIC tokens count as 1 byte). The command is not usable in direct mode.

```
10 TKEN "PING"
```

**UNTKEN** AV

The inverse of *TKEN*. Decodes the encoded string into its original form.

## STRUN AV

Executes, in program mode only, a string encoded with *TKEN* as if it were a BASIC line. The maximum length is 67 bytes. It allows the easy execution of formulae, or the creation of a command file.

All BASIC and DOS keywords except those normally not usable in direct mode may be included. Any string to be executed several times only needs to be tokenised once since it is stored in the nominated string variable.

```
10  A$= "PING"  
20  TKEN A$: STRUN A$      (Execute the PING)  
30  INPUT "EXPRESSION"; A$  
40  A$= "EP = "+A$  
50  TKEN A$: STRUN A$  
60  PRINT EP
```

✓ It is possible to execute a command normally restricted to direct mode within a program by using this command - e.g. *10 STRUN "B-"* to change the active drive. The only constraint is that if the string contains a BASIC keyword it must be tokenised with *TKEN* first. It is also possible to create an array of pre-defined commands, and to combine elements according to the context of the program. The possibilities are endless!

✓ A neat use of this command is to change drives within a BASIC program, a command otherwise lacking in SEDORIC:

```
100  PRINT "Drive (A/B/C/D)?": GETDR$  
200  IF DR$<"A" OR DR$>"D" THEN PING: GOTO 100  
300  DR$=DR$+"-": TKEN DR$: STRUN DR$: RETURN
```



## ERROR HANDLING ROUTINES

### **ERR SET or ERR OFF**

So that the programmer can handle errors generated by the DOS the *ERR SET* command prevents the interruption of a program by a DOS error. Note that errors generated by the BASIC interpreter cannot be handled by this method.

*ERR SET* is used in conjunction with the command *ERR GOTO*. When the DOS detects an error, the program continues to operate from the line specified by *ERR GOTO*, or will stop if the program doesn't provide such an error routine. An *ERR SET* must appear in the program before an *ERR GOTO*.

The variable EN at #4FD will contain the error number, and EL at #4FE-#4FF the number of the line causing the error. In this way, for example, the programmer can trap DOS errors generated by having the incorrect disc in the drive when looking for a specified data file. The error number 01 would provide evidence of this so the programmer could use an IF statement to print instructions to insert the correct disc. If an error is generated in direct mode, the line number will be given as 65535.

### **ERR GOTO NE**

Specifies the line number where DOS errors are handled provided an *ERR SET* has been executed.

N.B. - an *ERR SET* nullifies any previous *ERR GOTO*.

```
1000 ERR GOTO 10000
...
10000 PRINT "Enter a numeric variable !"
10010 GOTO 990
```

### **RESUME (NEXT)**

Restarts execution where it was interrupted (*RESUME*) or at the following command (*RESUME NEXT*). This command is used within the program section that handles the error condition.

### **ERROR NE**

Generates an error condition with the specified number held at location #4FD. This allows the centralisation of errors, including user-created ones. NE may range from 50 to 255. Numbers 1 to 49 are reserved for DOS errors.

If DOS error handling is enabled (by *ERR OFF*) then a 'USER XXX ERROR' message is generated, where XXX is the error number specified within the program.

*ERROR 50*

## TEXT FORMATTING

```
LINPUT (@NE, NE,) (AE,) NE; AV (, E) (, S) (, C) (, J) (, K)  
LINPUT (@X, Y, ) (Character, ) Length; AV...
```

An instruction to format text input - not numeric variables. It allows entry of text of the length specified, which is assigned to the string variable AV. This is a very powerful instruction which needs to be used progressively to realise its full potential. The simplest instruction is the form *LINPUT (length); AV*, it is an INPUT where you can select the number of characters to be entered.

The command creates a window on screen, which utilises a full page editor within the window. When inputting text using *LINPUT* only certain control characters are valid: CTRL-D, CTRL-T, CTRL-N, CTRL-Z (equivalent to ESC for attributes) and DEL. The cursor is automatically enabled.

The maximum window length is 255 characters. Input can be terminated by ESC or RETURN. The system variable OM (Out Mode) varies with the method of exit from the window:

0 = exit by RETURN	1 = ESC
2 = cursor left	3 = cursor right
4 = cursor down	5 = cursor up
6 = automatic exit (window full)	

- , **E** prevents erasure of the window before any text has been entered
- , **S** prohibits exit by use of the cursor arrows
- , **C** allows exit from the window when it is full without having to press RETURN. Without this option the cursor returns to the start of the window
- , **K** justifies the text in the screen window by inserting spaces, the variable does not contain the spaces
- , **J** the same as for , *K* except the variable is justified, but the screen unaffected.

The first character of AE fills the window if overwriting is required. If an empty string is set, the character is "." by default.

@ *X, Y* permits you to fix the position of the window on screen, as for PLOT.

```
LINPUT 20; A$, S  
LINPUT "-", 100; A$  
LINPUT @10, 3, 100; A$, C, J  
LINPUT @10, 3, "-", 2; A$
```

## CREATEW (FN)

Creates a screen area of 25 lines by 40 characters, starting at screen line two, for use by the *WINDOW* command. The command is not usable in Hires. A full screen editor is used within the screen area. CTRL-S saves the screen page, use CTRL-C to leave it.

Data fields are defined by CTRL-W, appearing as a square on the screen. The number of fields is unlimited, the field length is limited to 255 characters. The screen area can be thought of as a proforma for the entry of data into programs.

*CREATEW, "TEST"*

## WINDOW (FN)

Enables the loading of such text screen areas - not usable in Hires. If no filename is used (as saved by *CREATEW*) then the current screen area is loaded, if possible.

Before using this command, you must correctly dimension the array WI\$ with the number of fields in the screen area. If not, the default of 11 fields is used.

*WINDOW* displays the screen area to the screen along with the data fields and their values found in the array WI\$. The data is correctly justified on screen.

During data entry, the user can only write in the fields, or move from one field to another with the cursor arrows or RETURN. Most control characters are filtered out.

On typing CTRL-C all the fields are read and loaded in the WI\$ order (right to left, top to bottom). The number of fields allowed are unlimited, but WI\$ must be dimensioned correctly.

A field can be 255 characters long. If the Oric is in 40-column mode the fields lose their 38-column mode formatting. This problem also occurs on the Oric-1.

*WINDOW "TEST"* calls the TEST screen area into the buffer and displays it  
*WINDOW* displays the screen area currently in the buffer

## FORMATTED SCREEN/PRINTING

### **USING NE, AE (, AV)**

This command formats numbers printed to the screen; the number is the numeric variable NE, its format is defined in the string variable AE. If an alphanumeric variable is specified (,AV), the number formatted is not displayed, but the variable specified is affected accordingly.

The formats are given below; all other characters are unaffected:

- +** displays the sign of the number, + or -
- displays a space or the - sign if a negative number
- ↑** displays the exponent in scientific notation (e.g. +12)
- &a** is the character to replace 0's in front of a whole number, the default is spaces
- %x** is from 0 to 9, displays x characters of the whole number
- #x** is from 0 to 9, displays x characters of the decimal part
- !x** is from 0 to 9, rounds to x characters of the whole part
- @x** is from 0 to 9, rounds to x characters of the decimal part.

10 *A=1000/6 : USING A, "&0Number:%9.#4 E↑ "*  
displays Number: 000000166.6666 E+00

30 *USING A, "@2+%5. Pounds and #2 pence."*  
displays 1666 Pounds and 67 pence

40 *USING PI, "&\*%2 #5", A\$: PRINT A\$*  
displays \*3.14159

### **LUSING NE, AE (, AV)**

As for *USING*, but outputs to the printer.

*LUSING 12, "+-↑ #3"*

### **WIDTH (NE) or WIDTH LPRINT (NE)**

Sets the usable screen width (*WIDTH*) or printer width. NE is the number of characters per line to be printed before a carriage return (CR) and line feed (LF) are automatically generated.

On the Oric-1 *WIDTH* and *WIDTH LPRINT* each control both screen and printer and set them both to the value entered.

*WIDTH* alone resets the default values of 40 screen/80 printer on the Atmos, and 53 screen/93 printer on the Oric-1.

*WIDTH 20* gives a display 20 columns wide  
*WIDTH LPRINT 132* gives a printout 132 columns wide

## PRINTER OUTPUT

### **OUT NE**

Sends an ASCII code to the printer and is the equivalent of `LPRINT CHR$(NE)`. It is particularly useful for avoiding the bugs in the Oric-1 `LPRINT` command.

*OUT 27: OUT 33: OUT 30* equals `LPRINT CHR$(27);"!";CHR$(30)`

### **PR SET or PR OFF**

Switches the printer in or out. After a *PRSET* all screen output is directed to the printer.



*PRSET* treats all `PRINT`s as `LPRINT`s. *PROFF* does not treat `LPRINT`s as `PRINT`s!



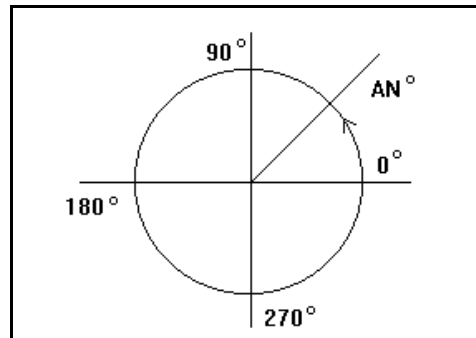
The printer must be disabled before executing a `STOP` or `END` in an Atmos program, or using `CTRL-C`. Otherwise a bug in the Atmos ROM sets the screen width incorrectly. A *WIDTH* command could be used to correct the fault. However this bug can be put to good use if you require double line spacing on your screen.



Because the *PR* routine in *SEDORIC* looks by itself for a *SET* or an *OFF*, and does not use the standard command interpreter, a bug has crept in. Do not use a space character after *SET* or *OFF*, or a syntax error is generated. Equally you must follow a *PR SET* or *PR OFF* inside an *INIST* string with a colon.

## GRAPHICS INSTRUCTIONS

These commands are constructed around the variable AN, representing the angle to be used. AN is expressed in degrees anticlockwise from EAST. Clockwise values may be expressed as a negative number of degrees.

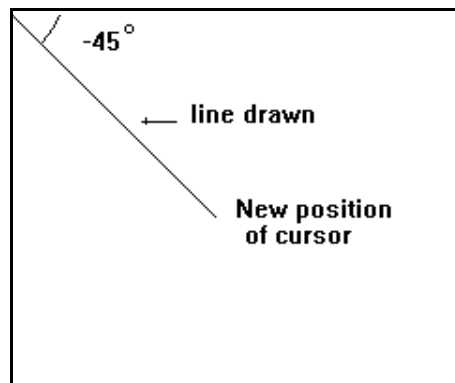


**LINE NE, FB code**

Draw NE points in the current direction, i.e. the current value of AN. FB code is as usual,  
FB=0 for current paper colour  
FB=1 for current ink colour  
FB=2 for inverse.

```
10 HIRES  
20 CURSET 0,0  
30 AN=-45  
40 LINE 100,1
```

draws a line from the top left hand corner of the screen 100 pixels long and at an angle of 45 degrees below the top of the screen (i.e. 45 degrees below (minus) EAST):



This command and *BOX* use #BFE0-#BFFF in RAM

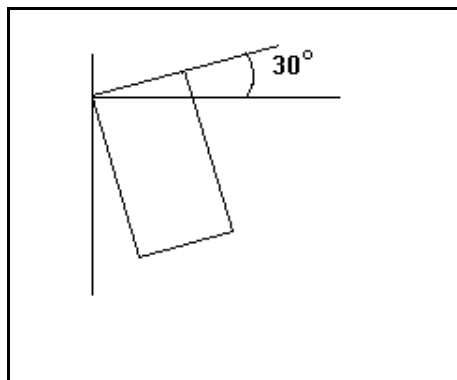
**BOX**  $NE_1$ ,  $NE_2$ , FB code

Draws a rectangle, with the current cursor position forming the top left corner. It will be angled to the current value of AN.

$NE_1$  and  $NE_2$  are the sides of the rectangle in pixels, across then down. FB code is as for *LINE*.

```
10 HIRES: PAPER 7: INK 0
20 CURSET 40, 40, 0: AN=30
30 BOX 50, 120, 1
40 CURSET 40, 20, 0: AN=-90
50 LINE 160, 1
60 CURSET 40, 40, 0: AN=0
70 LINE 150, 1
```

draws the following:

**LCUR**

Returns the horizontal and vertical coordinates of the cursor in Text mode in the variables CX and CY.

```
10 CLS: PRINT: PRINT "test LCUR";
20 LCUR
30 PRINT CX, CY
gives: 12 2
```

**HCUR**

As above for the Hires screen.

## USER COMMANDS

**USER x , DEF address ( , O)**

**USER x ( , A NE<sub>1</sub>) ( , X NE<sub>2</sub>) ( , Y NE<sub>3</sub>) ( , P NE<sub>4</sub>)**

x may be NV, a numeric variable.

Allows the call of assembly language sub-routines, with the ability to pass parameters. x or NV is the number of the user routine and must be in the range 0 to 3.

*USER x , DEF* (the comma is essential) is used to define the routine execution address. If the ,O option is specified, the routine will execute in RAM overlay.

*USER x* then executes the routine at the specified address. The 6502 registers are loaded with the specified parameters (which are optional).

On returning from the routine, the variables RA, RX, RY and RP contain the values of the registers A, X, Y and P.

*USER 2, DEF #CCB0*  
*USER 1, DEF #4000,0*  
*USER 2, X12, Y20*  
*USER A, A100, X200, P#C1*

**| (right bracket)**

As for the "!" in non-disc systems, used to call machine code routines from BASIC. DOKE addresses #2F9-#2FA with the call address

N.B. After a *QUIT*, the exclamation mark is obligatory to execute DOS commands.



## OTHER COMMANDS

### QUIT

Resets the pointers used by the DOS, resetting the IRQ and NMI vectors. Disables the FUNCT key, and makes the use of "!" obligatory for DOS commands.

The instruction is necessary before running any program that uses Page 4 of memory or modifies the IRQ/NMI vectors itself.

✓ This command provokes the question of how to return to SEDORIC after a *QUIT*? There is no command available other than a rather brutal *!RESET*, which doesn't always work. It would be interesting to program such a command - RETRIEVE perhaps?

### RESTORE (No. of line)

Places the DATA pointer at the start of the program or at the specified line (which does not have to exist). Only *!RESTORE* is treated by *RENUM*.

*RESTORE* (original BASIC command)  
*!RESTORE 100* (to differentiate the DOS command from the BASIC one)

### MOVE $AE_{start}$ , $AE_{end}$ , $AE_{target\ address}$

Moves a block of memory. The inclusive start address and end address+1, and target start address must be specified. Addresses above #C000 are in RAM overlay where the DOS is situated. Beware of corrupting this area.

*MOVE #A000, #BF40, #1000* saves the HIRES screen at #1000.  
*MOVE #1000, #2F40, #A000* recalls it

✓ While you should beware of corrupting RAM overlay, the possibility does mean that you can alter it! This has endless possibilities for programmers.

## RESET

A command having the same effect as the button on the disc drive.

## OLD

Recovers a BASIC program lost by a BOOT or a *NEW*. Clears all variables.

Sometimes when a program has completely disappeared, *OLD* will cause the Oric to hang up. Simply press the reset button.

Note that if variables have been affected between the *NEW* and the *OLD*, the program is not recoverable.

## RANDOM (NE)

Initialises the random number generator. If no argument is specified, the initialisation is left to chance (using the VIA timers). If one is specified, that number will always produce the same sequence.

*RANDOM 15*

## SWAP Variable, Variable

Exchanges the contents of the specified variables (of the same type).

*SWAP A, B*

## DATA FILES

SEDORIC DOS is capable of both sequential and random access to data files on disc. Each file has a logical number (LN) assigned to it to simplify the command syntax. The number may be from 0 to 63. It is necessary to open a file before accessing it, i.e. to allocate to it a logical number and associate it with a filename on disc. The DOS then loads all necessary information and creates the file on disc if it does not already exist. You can then use the file and access the data. On finishing with the file it is essential that it is closed.

N.B. if a 'WRITE-PROTECTED' or 'DISC FULL' error appears, it is advisable to close and then re-open the file for further access to ensure compatibility between disc and memory. Also all data files must be closed before a DIM instruction is used.

➡ The system of file generation uses a 'pseudo-array' placed right at the start of the BASIC array table area; it is an integer array, variable name FI. You must not use this variable name yourself or the system will crash on the first *OPEN*. Furthermore, after an *OPEN* and until the last *CLOSE* use of the BASIC command DIM is absolutely prohibited. If executed it overwrites the pseudo-array FI.

## SEQUENTIAL FILES

In sequential access files the data has to be read in the order in which it was written. Sequential operation is as if a pointer was moving along the file, advancing item by item. Reading or writing of data is carried out at the current pointer position. To enhance use of sequential files, instructions are provided to control the pointer position.

When writing to a file, data is immediately written to disc to minimise the risk of loss or corruption.

**OPEN S, FN, LN**

Opens a sequential file, reserves a memory buffer, and sets the pointer to start of file.

*OPEN S, "DISCS", 2* opens a file called "Discs" with the logical number 2.

If the file does not exist on disc, it will be created. Do not use the BASIC command DIM whilst a file is open.

**CLOSE (LN, LN, ...)**

Frees the buffer reserved by *OPEN* and closes any open files. Do not use without parameters in the middle of a program.

*CLOSE 34, 1* closes files 1 and 34

*CLOSE* closes all open files

**PUT LN, list of variables**

Writes the variable contents to a file.

*PUT 2, A\$, "TEST", 12, A%* writes in file 2 the variables A\$, TEST, 12 and the numeric variable A%

File size is only constrained by the free disc space. When writing to the middle of a file, the variable must be of the same type as that already written to the pointer position. With string variables, a string shorter than the previously-written string is packed with spaces to the right; a longer string is truncated from the right to the length of the previously-written string.

**TAKE LN, list of variables**

Reads from file LN the specified variables. The variables must be of the same type as those stored on the disc. If the end of file is reached during a *TAKE* operation an error message is generated.

**APPEND LN**

Places the file pointer at the end of the specified open sequential file.

*APPEND 2* places the pointer at the end of file no. 2

**REWIND LN**

Places the file pointer at the start of the specified open sequential file.

*REWIND 3*

**JUMP LN**, number of data items

Moves the pointer of an open sequential file the number of items specified. If the end of file is reached, acts as *APPEND*.

*JUMP 2, 200*  
*JUMP 12, A\*12*

**BUILD LN**

Enables you to build a sequential file by keyboard entry. The characters are entered on file as strings of 200 characters. After each 200 characters the string is saved to disc automatically. Exit with CTRL-C, when the characters input to the current 200 character string are saved to disc.

Code ASCII 13 (CR or RETURN) is automatically replaced by the code 13/10 (i.e. a line feed is added) to improve legibility when read with the *TYPE* command. All characters are faithfully reproduced on file, which can be useful for animation on the text screen.

*BUILD 2*

✓ Note that the first thing a *BUILD* command does is to execute an *APPEND*. Thus if the file is not empty, what you type will be added to the end of the file, even if you execute a *REWIND* beforehand. This can produce real oddities, so the moral is - use a new file each time with *BUILD*, unless you really want to append.

**TYPE LN**

Allows you to display the contents of an open sequential file, starting at the current position of the file pointer. Pressing a key will stop the listing, use SPACE to restart, and CTRL-C to exit.

This command can be used with *BUILD* for animated sequences.

*TYPE 2*

**LTYPE LN**

As for *TYPE*, but on the printer.

*LTYPE 2*

**&(LN)**

Returns -1 (true) if at the end of file, otherwise 0 (false).

N.B. the brackets are part of the command syntax, not an option.

*1000 A=&(2)  
1010 PRINT A*

**&(-LN)**

Returns the type of the next data item to read.

The code returned is:

0 for a numeric variable  
128 for a string variable  
-1 for end of file.

*A=&(-2)*

## RANDOM ACCESS FILES

Random access is very different from sequential access in that the data is organised within the FILE into RECORDS. Each record contains all the relevant information on the item recorded, and each has its own identity so that it can rapidly be accessed on disc.

Each record contains all relevant information grouped in FIELDS.

The record is the entity that is transferred from the disc to memory and vice-versa. To facilitate file operations, the commands work on a memory buffer holding the whole of the relevant record.

There are two types of instructions, those concerned with transfer between disc and buffer, and those with transfer of variables to and from the buffer.

**OPEN R, FN, LN (, record length, number of records reserved)**

Opens a file. It is essential that you specify the record length and number of records when first opening a file, but not necessary thereafter.

If the number of records exceeds the initial setting, extra space is automatically created for extra records.

The space occupied by a random access file is indicated in the number of sectors used on disc (i.e. number of records x record length / 256). A file of 300 records of 50 characters each would occupy about 60 sectors.

*OPEN R, "DIRECT", 2, 183, 100*

opens a file called "Direct", logical number 2, with each record a maximum 183 characters, and with space for 100 records reserved.

➡ DIM must never be used after an *OPEN* until a *CLOSE* has been executed.

**CLOSE (LN, LN ...)**

Closes the file numbered LN (or all files) and frees the buffer. You are recommended to use LN numbers for the greatest security.

*CLOSE 1*

## TRANSFERS BETWEEN BUFFER AND DISC

**TAKE LN, Record no.**

Loads the specified record from disc into the buffer.

*TAKE 2, 120* (i.e. No. 120 of logical file number 2)  
*TAKE F, JP\*12*

If the record does not exist a 'BAD RECORD NUMBER' error will be generated.

*TAKE 2, 120*  
*TAKE F, JP\*12*

**PUT LN, Record no.**

Transfers the specified record from the buffer to the file on disc.

If the record does not exist, it will be created. It is preferable to create sufficient records to start with, as adding an additional record to the file on disc is a comparatively lengthy process.

*PUT 2, 120*



## WORKING ON THE BUFFER

To facilitate the classification of data, the record can be further divided into fields, enabling the place within the record where the data will be held to be defined. The fields are rather like variables; you may give them a value and read them as with variables. However be warned, they are not variables!

The conventions for field names (FLN) are :

- 5 significant characters
- can contain a pseudo array by indexing the field
- the field name with index 0 is equivalent to the name without an index, i.e. if a field is called NAME, NAME(0) is the same as NAME
- must not include a BASIC keyword or start with a DOS keyword
- the total number of fields is only limited by the available memory.

E.g.: *NAME*  
*INDEX (1)*  
*ADDRESS1 (0)* equals *ADDRE (0)*, only the first five characters are significant.

<b>FIELD LN, FLN TO type (,....)</b>
--------------------------------------

Defines fields within the each record of the file. The type of field is defined by:

**\$NE** for an alphanumeric field, NE defines its length  
**%** for an integer field  
**!** for a numerical field  
letter **O** for a field of one byte

If the command ends with a comma, the next definition starts at the current position. The length of a field must not exceed the total length of the record.

An integer field is 2 bytes long, a numerical 5 bytes, and an alphanumeric the length specified. To that is added 2 bytes for internal working space.

*FIELD 2, NAME TO \$8, SURNAME TO \$12, AGETO %, SALARY TO !, SEX*

opens in file number 2 a field for the name of 8 characters, one of 12 characters for the surname, an integer field for the age, a numeric field for salary, and a single byte field for the sex.

Total length of record:  $2+8 + 2+12 + 2+2 + 2+5 + 2+1 = 38$  bytes per record

➡ Note the bug and solution dealt with on page 58.

## TRANSFERS BETWEEN FIELDS AND VARIABLES

**RSET** FLN < Expression

Writes the given value to a field - both must be of the same type.  
String variables will be justified to the right, and stripped at the left if the string is too long for the field, any gap to the left is filled with spaces

*RSET NAME(2) < "EUREKA"* (NAME(2) is an alphanumeric field)  
*RSET TEST < 123* (TEST is an integer field)

The '<' is part of the syntax of this command. If the field STRING has five characters:

*RSET STRING < "123456"* stores "23456"  
*RSET STRING < "TOTO"* stores " TOTO"

**LSET** FLN < Expression (,.....)

Identical to *RSET* for numeric and integer fields. String variables are justified to the left and stripped at the right if too long, or completed at the right by spaces if too short.

*LSET TEST < 2000\*AS*

If the field STRING has five characters: *LSET STRING < "TOTO"* stores "TOTO "  
*LSET STRING < "123456"* stores "12345"

**FLN** > Variable

Read field of name FLN and put into the specified variable, which must be of the same type.

*NAME(1) > A\$*

*TEST > A*

*PROPERTY > A* is invalid because *PR* is a DOS keyword.

➡ This command has no in-built security. It should not be used with filetype ,S, yet this is not tested. There is no test as to whether a ,R type file is open, nor whether a record has been loaded with *TAKE*. The validity of the data loaded is not checked. The moral is - use with care!

**&** (LN)

Display the number of records in file number LN

*& (12)*

✓ This command cannot be used with 'D' type files. Used with 'R' type files, it returns the number of records with *&(+no.)*, the length of record with *&(-no.)*.

## DISC ACCESS

There is a special kind of file which allows you to modify the disc, sector by sector, directly from BASIC. It enables the experienced programmer to create a filing system particular to the needs of the application. The idea is similar to the instructions for working on the buffer files, but the fields are here defined without the 2 bytes working space.

Instead of the record being in the buffer, the buffer now contains the contents of the disc sector. The buffer is 256 bytes long.

To ease things, instructions are included to search for empty sectors on the disc.

**OPEN D, LN (,Drive)**

Reserves the buffer of 256 bytes and opens the file FLN. Note that the space between *OPEN* and the *D* is obligatory to avoid the BASIC keyword END.

*OPEN D, 1*  
*OPEN D, 1, B*

**CLOSE (LN (, LN.....))**

Closes the file(s) specified and frees the buffer.

**TAKE LN, Track, Sector (,**

Reads the specified sector into the buffer. No test is carried out to see if the specified sector exists. If it does not, a 'DISC I/O (ERROR number 04)' message is generated.

*TAKE 2, 14, 3, A*  
*TAKE 0, 20, 1*

**PUT LN, Track, Sector (, Drive)**

As *TAKE*, but used to write, not read the sector.

For a double-sided drive, add 128 to the track number to access side 2 of the disc:

*PUT 1, 131, 2* writes the buffer in sector 2 of track 3 on Side B

☞ Be aware that for this use of the *PUT* command, there is no verification of the validity of the track and sector numbers entered.

## SECTOR RESERVATION

It is possible to search for free sectors on the disc, as well as to free certain sectors. All this information is stored in special sectors (Track 20, Sectors 2 & 3) called **BITMAP**. To reserve a sector it is necessary to read **BITMAP**, determine the free the sector using *CRESEC*, and rewrite **BITMAP**.

### **PMAP Drive**

Reads the **BITMAP** into memory.

*PMAP*  
*PMAP A*

### **SMAP Drive**

Writes to disc the **BITMAP** in memory.

Between a *PMAP* and an *SMAP* you must not do a *LOAD*, *SAVE* or *DIR*, or make direct or sequential access to the disc, otherwise **BITMAP** may be corrupted.

*SMAP A*

### **CRESEC**

Returns in the system variables *FP* and *FS* the track and sector address of a free sector. If the disc is full, it says so.

It is only usable after a *PMAP*. and automatically decrements the number of free sectors.

### **FRSEC** $NE_{\text{track}}, NE_{\text{sector}}$

Frees the specified sector and increments the number of free sectors. If already free, it has no effect.

Only usable after a *PMAP*.

*FRSEC 20, 10*

## INDEX OF DOS KEYWORDS

Command	Page	Command	Page	Command	Page
ACCENT	20	INSTR	27	RESUME	29
APPEND	40	JUMP	41	REWIND	40
AZERTY	19	KEY	19	RSET	46
BACKUP	12	KEYDEF	25	SAVE	8
BOX	35	KEYIF	19	SAVEM	8
BUILD	41	KEYSAVE	26	SAVEO	8
CHANGE	17	KEYUSE	25	SAVEU	8
CHKSUM	5	LCUR	35	SEARCH	6
CLOSE	39-43-47	LDIR	6	SEEK	16
COPY	12	LINE	34	SMAP	48
CREATEW	31	LINPUT	30	STATUS	10
CRESEC	48	LOAD	7	STRUN	28
DEL	9	LSET	46	SWAP	38
DELBAK	9	LTYPE	42	SYS	13
DELETE	18	LUSING	32	SYSTEM	14
DESTROY	9	MERGE	16	TAKE	40-44-47
DIR	5	MOVE	37	TKEN	27
DKEY	14	NUM	18	TRACK	14
DNAME	13	OLD	38	TYPE	41
DNUM	14	OPEN	39-43-47	UNPROT	10
DSYS	13	OUT	33	UNTKEN	27
DTRACK	14	PMAP	48	USER	36
ERR	29	PR	33	USING	32
ERRGOTO	29	PROT	10	VIEWHIRES	5
ERROR	29	PUT	40-44-47	VUSER	26
ESAVE	8	QUIT	37	WIDTH	32
EXT	4	QWERTY	19	WINDOW	31
FIELD	45	RANDOM	38	]	36
FRSEC	48	REN	9	>	46
HCUR	35	RENUM	15	&	42-46
INIST	13	RESET	38	<	46
INIT	11	RESTORE	37		

## APPENDIX - SEDORIC DOS Keyword Addresses and Codes

Command	Code	Entry	Block	Command	Code	Entry	Block
		Address				Address	
ACCENT	35	EB91		LUSING	80	F036	
APPEND	33	FE07		MERGE	88	F13C	3
AZERTY	34	EBDE		MOVE	87	F136	1
BACKUP	37	F151	2	NUM	89	EB25	
BOX	36	F0DE		OLD	91	E0AF	
BUILD	38	FEE0		OPEN	92	FA50	
CHANGE	39	F148	3	OUT	90	E71F	
CLOSE	40	FB8D		PMAP	96	F990	
COPY	41	F157	4	PR	95	E7C0	
CREATEW	42	DE4D		PROT	94		7
CRESEC	43	F9BC		PUT	93	F9CB	
DEL	48	E446		QUIT	97	E7F5	
DELBAK	47	E437		QWERTY	98	EBE1	
DELETE	45	F142	1	RANDOM	105	E796	
DESTROY	46	E444		REN	103	E537	
DIR	49	E344		RENUM	102	F14E	1
DKEY	53	F124	5	ESET	100	E7B8	
DNAME	52	F145	5	RESTORE	106	E7D9	
DNUM	51	F12A	5	RESUME	99	E9BB	
DSYS	54	F127	5	REWIND	101	FABB	
DTRACK	55	F139	5	RSET	100	FC75	
ERR	60	E97F		SAVE	117	DD50	
ERRGOTO	57	E999		SAVEM	115	DD4A	
ERROR	59	E9B0		SAVEO	116	DD53	
ESAVE	61	DDE0		SAVEU	114	DD4D	
EXT	62		7	SEARCH	118	E5FC	
FIELD	63	FBBF		SEEK	109	F154	3
FRSEC	64	F99C		SMAP	120	F996	
HCUR	65	EBF5		STATUS	113		7
INIST	68	F12D		STRUN	111	E853	
INIT	66	F169	6	SWAP	108	EA3B	
INSTR	67	EC2E		SYS	119	F159	5
JUMP	69	FE12		SYSTEM	112		7
KEY	76	E70B		TAKE	122	F8DF	
KEYDEF	73	D9FD		TKEN	121	E89D	
KEYIF	71	DA20		TRACK	124	F130	5
KEYSAVE	75	DDCD		TYPE	123	FE98	
KEYUSE	72	D9B0		UNPROT			7
LCUR	86	EBEC		UNTKEN	126	E8E1	
LDIR	84	E7D0		USER	125	EA7F	
LINE	77	F079		USING	127	EE99	
LINPUT	82	EC94		VUSER	129	F121	5
LOAD	83	DF7		WIDTH	130	E740	
LSET	78	FC73		WINDOW	131	F21C	
LTYPE	85	FE95		]		ECo4	

The block reference refers to the external blocks of the DOS that have to be loaded from disc when the command is used for the first time.

The entry point addresses (in hexadecimal notation) can be used in conjunction with the *MOVE* command to examine the DOS and customise it if desired.

## APPENDIX - Error Messages

### **01 - FILE NOT FOUND**

The file has not been found on the disc (the file may not exist, the filename may be misspelt or a wildcard misplaced).

### **02 - DRIVE NOT IN LINE**

A specified drive is not connected. Check the drive number. To connect it use the *SYS* command.

### **03 - INVALID FILE NAME**

The filename contains invalid characters or is too long (9 characters maximum).

### **04 - DISC I/O**

The disc is corrupted. The DOS displays the sector and track where the error was detected to assist experienced users to recover the disc (e.g. using the NIBBLE disc sector editor). Note: The sector number is not significant, nor is it displayed, if an error occurs during formatting of a disc.

### **05 - WRITE PROTECTED**

Indicates an attempt to write to a disc which has the write-protect tab set.

### **06 - WILDCARD(S) NOT ALLOWED**

A wildcard has been used in a filename in connection with a command that does not accept wildcards (e.g. *LOAD*).

### **07 - FILE ALREADY EXISTS**

An attempt has been made to save a file on a disc which already contains a file of that name.

### **08 - DISC FULL**

The file is too long to be saved in the available disc space.

### **09 - ILLEGAL QUANTITY**

A parameter has been specified that is too large or too small.

### **10 - SYNTAX ERROR**

A mistake has been made in typing an instruction.

### **11 - UNKNOWN FORMAT**

An attempt has been made to use a disc that has not been formatted by the current version of SEDORIC DOS.

### **12 - TYPE MISMATCH ERROR**

An attempt has been made to match one type of variable or field against another, such as a numeric against a string variable.

### **13 - FILE TYPE MISMATCH**

A file of one type has been specified with a name associated with a file of another type (e.g. attempting to load data with the *LOAD* command).

### **14 - FILE NOT OPEN**

An attempt has been made to access a file that has not been opened.

### **15 - FILE ALREADY OPEN**

An attempt has been made to assign a logical number to a file when that number has already been assigned to another file.

**16 - END OF FILE**

An attempt has been made to read data when the end of the file has been reached.

**17 - BAD RECORD NUMBER**

The record number entered exceeds the capacity set for the file.

**18 - FIELD OVERFLOW**

The field definitions have produced an overall length that is greater than the length specified for the record.

**19 - STRING TOO LONG**

The string entered is too long (see *TKEN*, etc.).

**20 - UNKNOWN FIELD NAME**

An attempt has been made to read from or write to a field not previously defined with the **FIELD** command.

APPENDIX - Standard Function Key Usage

<u>Code</u>	<u>Definition</u>	<u>Code</u>	<u>Definiton</u>
000-015	User definable		
000	(space character)	016	HEX\$(
001	DOKE#2F5,#	017	CALL#
002	DOKE#2F5,#467 + RETURN	018	TEXT + RETURN
003	DOKE#2F9,#	019	FORI=1TO
004	DOKE#2F9,#Do70 + RETURN	020	LEFT\$(
005	DOKE#2FC,#	021	MID\$
006	DOKE#2FC,#461 + RETURN	022	RIGHT\$
007	PAPER0:INK7 + RETURN	023	STR\$(
008	CALL#F8Do + RETURN	024	UNPROT + RETURN
009	ê	025	©
010	?HEX\$(PEEK(#	026	USING
011	?HEX\$(DEEK(#	027	VIEWHIRES
012	PEEK(#	028	VUSER + RETURN
013	DEEK(#	029	WIDTH
014	POKE#	030	WINDOW
015	DOKE#	031	!RESTORE
032 to 127	DOS keywords (see page 23)		
128 to 253	BASIC keywords (see page 23)		
254	DEL		
255	Generate line numbers.		



## APPENDIX - System Variables

These variables are used by the DOS in conjunction with the relevant command or error condition. They can be read from BASIC programs, but should not be assigned new values from within the program if the associated DOS command is to be used, except for the AN variable.

### **Used by SEARCH:**

**EF** (Existing File) EF=1 if the file exists, 0 if it doesn't.

### **Used by LOAD or direct loading:**

**ST** (STart address) contains the start address of program

**ED** (EnD address) contains the end address

**FT** (File Type) contains the program type

**EX** (EXecution address) of the program.

Note that these variables will be cleared if a BASIC program is allowed to run on loading. they may also be corrupted if a file is loaded into the area of memory occupied by the variables. To preserve them, use the ,V option which will only load in the status byte and not the whole file.

### **Used by SEEK:**

**SK** (SeeK) contains the number of string occurrences sought.

### **Used by INSTR:**

**IN** (INstring) contains the sub-string position in the string

### **Used by the error handling routines:**

**EN** (Error Number) displays the DOS error number

**EL** (Line number) displays the line where the error is.

```
100 ERRGOTO 1000
```

```
.....
```

```
1000 PRINT "Error"; EN, "in line"; EL
```

### **Used by LINPUT:**

**OM** (Out Mode) returns the mode of exit from entering text into the window buffer.

### **Used by WINDOW:**

WI\$ is an array containing the number of fields used within the specified window. Its default is 11 as in BASIC.

### **Used by LINE and BOX:**

**AN** (ANgle) contains the angle in degrees used in the commands *LINE* or *BOX*. Default is 0 degrees. This must be written to from a BASIC program.

### Used by HCUR and LCUR:

**CX** contains X coordinate of text or graphic cursor  
**CY** contains Y coordinate of cursor.

### Used by USER:

**RA** (Register Accumulator)  
contains the value of the A register of the 6502  
**RX** (X Register)  
contains the value of the X register  
**RY** (Y Register)  
contains the value of the Y register  
**RP** (Processor status Register)  
contains the value of the Status register

### Used by CRESEC:

**FP** (Free Piste [Track])  
contains the track number of the freed sector  
**FS** (Free Sector)  
contains the sector number.

### Used by Data Files:

**FI** is a pseudo-array created when a data file is opened. It is stored at the start of BASIC arrays, and is overwritten if a DIM command is used while a file is open.

## APPENDIX - Disc Structure

The first 94 sectors of a Master disc (from Sector 1 of Track 0) are occupied by the boot routines and then the DOS.

The sectors identified below have a fixed position and are reserved at the time of the disc initialisation.

<u>Track</u>	<u>Sector</u>	<u>Use</u>
20	01	Disc name
20	02	Bit map
20	03	Bit map
20	04	Directory 1
20	07	Directory 2
20	10	Directory 3
20	13	Directory 4
20	16	Directory 5

The remaining sectors are free for the storage of files or data.

## APPENDIX - File Status Byte Coding

The status byte of a file is binary coded and the significance of each bit is given below. The byte follows the standard bit format, i.e. b7 to b0. The bit is active if it is set to a '1' and inactive if it equals '0'.

b0 : automatic execution  
b1 : unused  
b2 : unused  
b3 : direct access  
b4 : sequential access  
b5 : window (note - b6 = 1 also)  
b6 : data block  
b7 : BASIC file

## APPENDIX - Keyboard Codes for KEYIF Instruction

Key	Code	Key	Code	Key	Code
1	#A8	2	#B2	3	#B8
4	#9A	5	#90	6	#8A
7	#80	8	#87	9	#8B
0	#97	-	#9B	=	#BF
\	#B3	ESC	#A9	Q	#B1
W	#BE	E	#9E	R	#91
T	#89	Y	#86	U	#85
I	#8D	O	#95	P	#9D
[	#BD	]	#B5	DEL	#AD
CTRL	#A2	A	#AE	S	#B6
D	#B9	F	#99	G	#96
H	#8E	J	#81	K	#83
L	#8F	;	#93	"	#BB
RETURN	#AF	SHIFT L	#A4	Z	#AA
X	#B0	C	#BA	V	#98
B	#92	N	#88	M	#82
,	#8C	.	#94	/	#9F
SHIFT R	#A7	→	#AC	↓	#B4
SPACE	#84	↑	#9C	←	#BC
FUNCT	#A5				

## APPENDIX - Switching between RAM and ROM

The DOS is contained in RAM overlay, which 'shadows' the BASIC ROM.

When in ROM, **JSR #04F2** takes you to RAM overlay. Another **JSR #04F2** returns control back to the ROM.

When using this routine there is a risk of 'hang-up' when returning to BASIC so care must be taken. Switching between ROM and RAM overlay does not affect any of the 6502 registers.

## APPENDIX - Utility Files on SEDORIC Master Disc

The SEDORIC master disc contains several programs and utilities.

### **CONVERT**

Easily transfers your files created under ORIC DOS format to SEDORIC. Load by typing CONVERT. The program is menu-driven - move between choices with SPACE or cursor arrows, and validate a highlighted choice with RETURN.

ESC returns you immediately to the menu, and ESC at the menu returns you to BASIC. There are 4 menu options:

#### Selection of drives

Drive selection - enables you to select which drive will hold the old format disc and which the new - they can of course both be the same drive, as with the *COPY* command.

Select the required drive with the SPACE BAR and press RETURN to toggle between the source and target drives.

ESC to return to the menu.

#### Initialisation of target disc

Enables you to format and initialise the target disc, equivalent to the command *INIT*. A slave disc is formatted. Since the track and sector format is read from memory the values default to those read from the bootup DOS disc. Therefore double-sided formatting of discs of 80 tracks is supported.

#### Selection of the DOS version

Always leave this on ORIC DOS - the others are French DOS systems.

#### Conversion of files

The heart of the utility. Insert the source disc in the relevant drive. Press the DEL key to produce a catalogue of files on the source disc - only the first 80 (!) files are copiable.

To select files to transfer/convert, move the highlight bar with the cursor arrows to each file that you wish to transfer in turn, and at each file press SPACE. A star will appear by each file you select. When you have finished selecting files, press RETURN to start the file conversion. If one drive is being used, the usual prompts for loading source and target discs will appear. As each file is transferred a '+' replaces the '\*'.

If an error occurs, press ESC to move to the next file, or any other key to retry the file that produced the error.

Note: CONVERT will copy ORIC DOS data files across onto the target disc, but they will not be converted for use by SEDORIC DOS.

### **GAMEINIT**

To initialise a games disc containing 'SHORTSED' - a DOS of only three instructions, *!LOAD*, *!DIR* and direct file loading (i.e. *!(name of file)*), each with its usual syntax. This disc will have only 17 sectors occupied by the DOS, as against 100 plus for a normal Master disc.

The syntax is close to that of *INIT*, except that you put a colon after the command, e.g. *GAMEINIT:A,17,42*. If no parameters are entered, the default values of 42 tracks, 17 sectors, drive A are used.

NOTE: *GAMEINIT* does not permit more than 1919 sectors on a double-sided disc.

## **ROMORIC1**

This loads the V1.0 ROM into RAM overlay, giving you the Oric-1 ROM on an Atmos. Note of course that you no longer have the DOS in memory, and any programs must be loaded from/saved to cassette.

## **ROMATMOS**

The identical facility to give the Oric-1 a V1.1 ROM in RAM overlay.

## **ADDRESS**

This is an example of a direct access file. Type ADDRESS and press RETURN to run the program. There are two associated files - ADDRESS.WIN and ADDRESS.DAT

## **ALPHA**

ALPHA (RETURN) enables you to sort a disc's directory in alphabetical order. It allows a disc to be sorted in any drive.

## **STAT**

STAT enables you to produce statistics on the number of times instructions are used in a BASIC program. Type *STAT, J*, then RUN 64000. The program only lists those commands actually encountered.

## **VERSION**

Gives you the version of SEDORIC you are using. It allows a disc to be inserted in any drive.

## **SECTOR MAP**

This utility displays the state of all sectors on the disc, whether they are (O)ccupied or free (.). Output can be to the screen or printer. To run type SECTMAP and press RETURN. Discs formatted to 80 tracks double-sided are supported.

## APPENDIX - Known bugs and limitations

Version 1.006:

1. The use of lower case letters in SEDORIC commands is risky - in a number of cases parameters simply will not work in lower case (the most obvious is AUTO). It is safer to stick to capital letters for DOS keywords and parameters.
2. SEDORIC overwrites the flag at #0E that CSAVE uses to indicate that an ,A parameter has been entered. All such saves are executed from #501 whatever value is entered with ,A.
3. KEYDEF is seriously bugged.
4. TAKE command - variables were corrupted.
5. LDIR does not filter out screen control codes.
6. FIELD command only operated on one random access file at a time.
7. Formatting a double-sided disc did not give the correct indication when displaying the directory.
8. In formatting a double-sided 80-track disc, the number of sectors was limited by the BITMAP to 1919. INIT was therefore limited to the following formats for 16 or 17 sectors per track: *INIT A, 16, 59, D*      *INIT B, 17, 56, D*
9. Format failures can occur when using INIT or BACKUP for formatting.
10. All utilities were written in French.

## APPENDIX - Developments and solutions

Version 1.007:

2. The only available solution for the CSAVE bug is to QUIT before using the command with ,A ,E parameters.
4. The TAKE command has been fully corrected.
6. FIELD bug: When opening more than one random access file at once, include the FIELD definition in a sub-routine and call it prior to accessing a particular file.
7. Double-sided directory display bug: After the first format return to command level, re-enter INIT, do not format, but complete the initialisation as required.
9. If a format failure occurs, there is no option but to run the command again.
10. All utilities are translated into English.

Version 2.0:

8. The restriction of the total number of sectors permissible (formerly 1919) has been removed.
9. The directory indication for a double-sided disc is now correct following the first format.
11. The *SECTMAP*, *ALPHA* and *VERSION* utilities on the master disc have been updated.
12. The packaged game has been changed from M.A.R.C. to Krillys.
13. The master disc contains a file **NIBFIX** which will amend the disc sector editor 'NIBBLE' to work with the updated DOS.
14. Some other V1.0 machine code programs are incompatible with V2.0

Version 2.1:

15. Minor bug fixes, including some created by V2.0 in Bank 5 (the DNAME and INIST commands).

Version 3.0:

1. The use of lower case letters in SEDORIC commands is now prohibited.
2. CSAVE works correctly.
3. KEYDEF works correctly and has been completed.
14. All machine code programs (V1.0 or V2.0) are compatible with V3.0