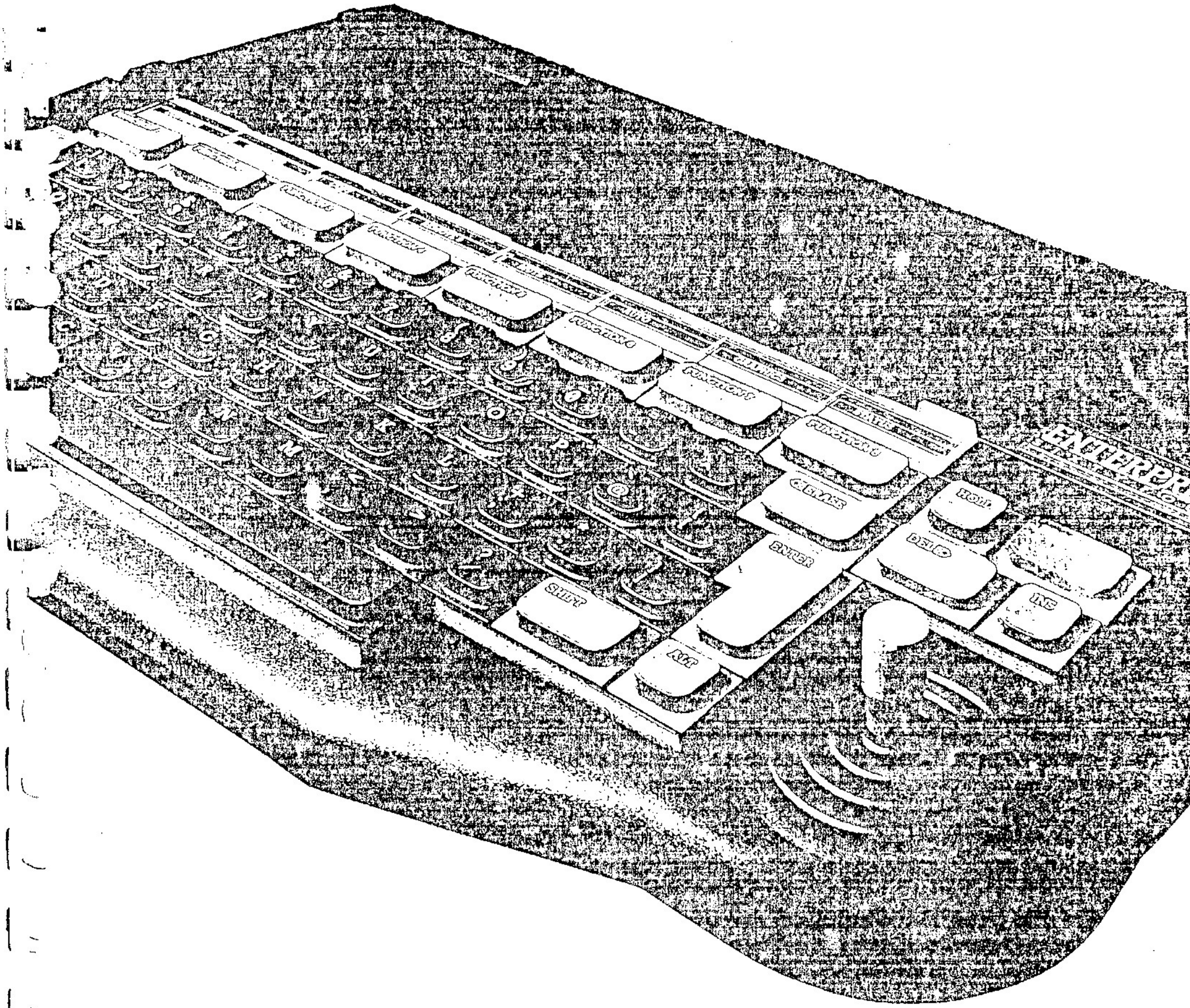
ENTERPRISE COMPUTERS



EXDOS 0.0

CONFIDENTIAL - INTERNAL CIRCULATION ONLY

Document PER-1
Title ..... EXDOS - System Overview
Issue 1
Date ..... 16th April 1985

CONTENTS

1. Introduction - EXDOS and IS-DOS
2. EXDOS Organisation
3. IS-DOS Organisation

EXDOS/IS-DOS is a relatively complicated system and so it is the intention of this document to give a very brief overview of how the various parts of the system relate to each other, in order to help in understanding the rest of the documentation. More details of the various parts of the system can be found in the following documents:

PER-2	EXDOS - DISKIO Specification
PER-3	EXDOS - Unit Handler Specification
PER-4	DISKIO and UNITH Implementation Notes
PER-5	EXDOS - System Specification
PER-16	IS-DOS - System Specification

1. INTRODUCTION - EXDOS and IS-DOS

It is important in understanding the system, to distinguish between EXDOS and IS-DOS.

EXDOS is a disk extension to the EXOS operating system, contained in ROM on the disk controller card, which is automatically linked in when the Enterprise is switched on. It provides an EXOS disk device and also various extension commands which are available to all applications programs through the normal "scan system extensions" EXOS call (for example "colon" commands from IS-BASIC).

IS-DOS is an EXOS applications program which can be loaded from disk either automatically when EXDOS starts up, or afterwards by an explicit command from the user. Once it is loaded, IS-DOS controls the machine in the much the same way that IS-BASIC normally does. It provides a command line environment similar to MS-DOS in which the user can type commands to IS-DOS or load transient programs which run and then return to IS-DOS.

2. EXDOS Organisation

The central part of EXDOS is the FILING SYSTEM HANDLER. This provides all the facilities necessary to access and manipulate files on disks. The user's applications program does not normally talk to the filing system handler directly but rather goes through two possible routes, the DISK DEVICE or the EXDOS CLI.

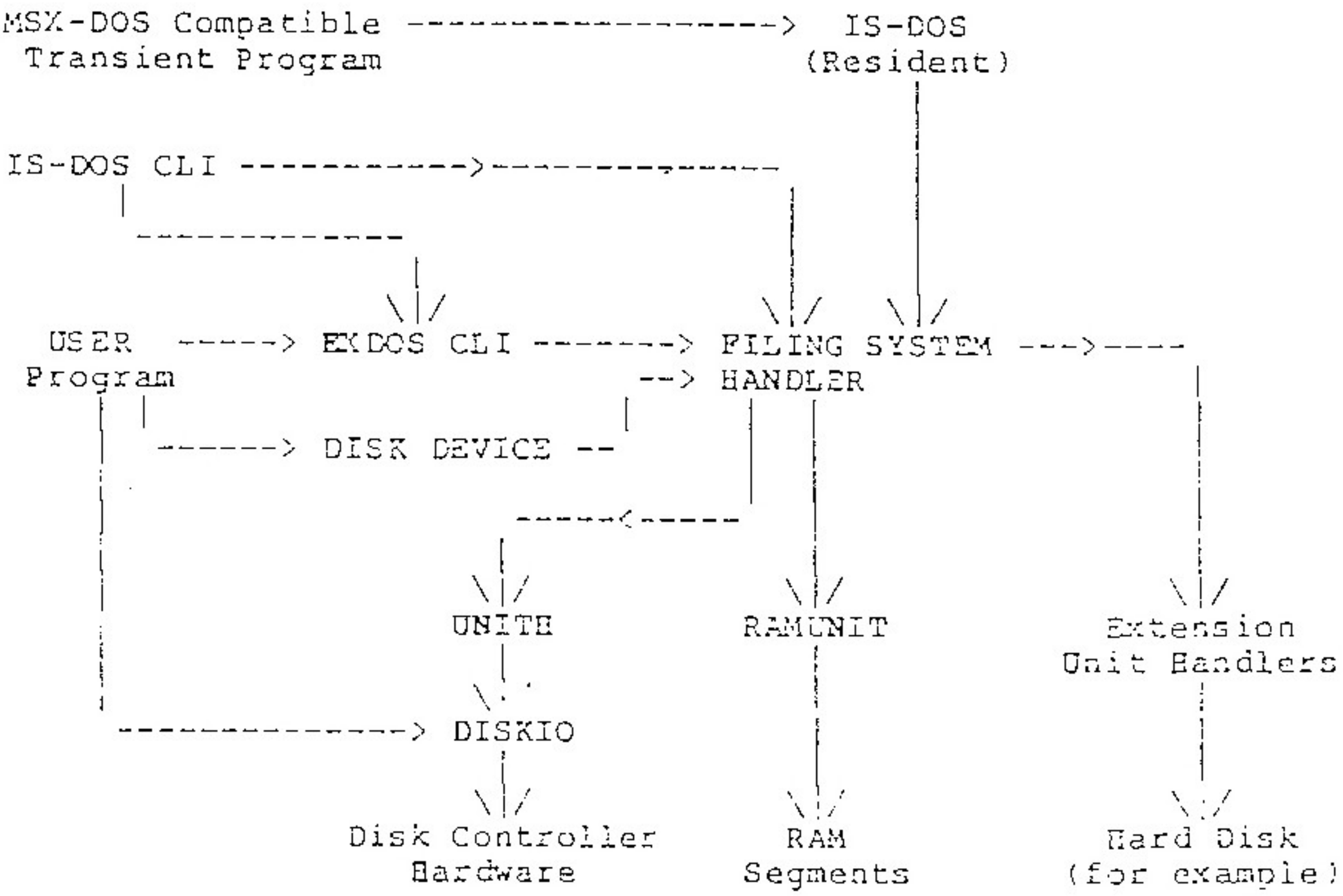
The DISK DEVICE is an EXOS device driver which EXDOS automatically links in when it starts up. This can be used like any other EXOS device, allowing the user to open and close channels to disk files, and read or write data to them.

The EXDOS CLI is an EXOS system extension which interprets certain command strings ("colon" commands from IS-BASIC) to provide functions such as renaming and deleting files and giving directory listings.

When EXDOS needs to access a disk, it always does so through a UNIT HANDLER. There is a built in unit handler (called UNITH) contained in the EXDOS ROM which provides an interface to up to four disk drives connected to the disk controller card. Additional unit handlers can be linked in to allow EXDOS to access any other disk hardware which may be created. In addition to UNITH, the EXDOS ROM also contains another unit handler (called RAMUNIT) which can be linked in by an EXDOS CLI command to provide a RAM-disk facility.

UNITH does not access the disk controller hardware directly. Instead, it always goes through DISKIO which provides a very low level interface to the disk controller. This low level interface is also available directly to the user for accessing non-standard disks, and also for functions like disk formatting.

The inter-relation of all these various sections of code is shown in the diagram below. IS-DOS is included in this diagram to show how it fits in, although it is not contained in the EXDOS ROM (see the next section). The USER in this diagram refers to EXOS applications programs such as IS-BASIC, and also to the person using these programs.



EXDOS System Organisation

3. IS-DOS Organisation

The code of IS-DOS is on disk,rather than being in ROM and will be loaded automatically at power on, or by an explicit EXDOS CLI command. When loaded it sets up a standard EXOS environment with certin defined channels open (for example video, keyboard and editor), and interprets commands typed by the user.

The way in which IS-DOS fits in with EXDOS is shown in the diagram above, although only the most important connections are shown. The user's commands are interpreted by a section of IS-DOS called the IS-DOS CLI. This handles commands in various different ways, the important ones of which are:

1. Passing them to the EXDOS CLI for interpretation.
2. Interpreting them internally, using the FILING SYSTEM HANDLER to carry out disk operations (and maybee using EXOS as well).
3. Passing them to the EXOS "scan system extensions" function call for interpretation by other ROMs.
4. Loading the ".COM" file from disk and executing it as a transient program.
5. Reading subsequent commands from the ".BAT" file

When IS-DOS loads a transient program, it provides it with a CP/M and MSX-DOS compatible environment in which to run. This is done by the resident portion of IS-DOS which remains in memory while the transient program is running (the IS-DOS CLI can be overwritten by the transient program). This resident portion translates CP/M function calls into calls to the FILING SYSTEM HANDLER, and also handles the re-starting (and possibly re-loading) of the IS-DOS CLI when the transient program finishes. This latter function includes controlling batch file processing.

Transient programs loaded by IS-DOS can make all important CP/M and MSX-DOS calls, and some CP/M BIOS calls. This ensures that programs written for CP/M or MSX-DOS will work. under IS-DOS. However IS-DOS programs can also make EXOS function calls and can thus make full use of all the Enterprise's features. These calls can be mixed together in any way.

++++++++++ END OF DOCUMENT ++++++++++

CONFIDENTIAL - INTERNAL CIRCULATION ONLY

Document PER-5
Title ..... EXDOS - System Specification
Issue 2
Date ..... 26th April 1985

Note: This document describes the ROM resident part of EXDOS which provides disk extensions to the EXDOS operating system. It does not describe the CP/M and MSX-DOS emulation facilities or the IS-DOS command environment, these are covered by a separate document (PER-16). Related documents are:

PER-1	EXDOS - System Overview
PER-2	EXDOS - DISKIO Specification
PER-3	EXDOS - Unit Handler Specification
PER-4	DISKIO and UNITH Implementation Notes
PER-16	IS-DOS - System Specification

CONTENTS
==========

1. Introduction
1.1 EXDOS ROM Entry Points
2. Initialisation
2.1 The Initialisation File - "EXDOS.INI"
2.2 Re-initialisation
3. The EXDOS Filing System Handler - FISH
3.1 FISH Function Calls - General
3.2 EXOS Variables
3.3 File Control Blocks (FCBs)
3.3.1 FCB Format
3.3.2 Pathname Strings
3.4 \
. \ Details of FISH function calls
. /
3.28 /
4. Filing System and Disk Allocation
4.1 Boot Sector
4.2 File Allocation Table
4.3 Directory Entry Format
4.3.1 File Attributes Byte
4.4 Volume ID
4.5 Dirty Disk Flag (Un-deletion)
5. Disk Buffering
5.1 Creating Additional Sector Buffers
5.2 Additional Permanent Buffers
6. Disk Device
6.1 Disk I/O Channels
7. EXDOS CLI Commands
7.1 Command Parameters
7.2 \
. \ Details of EXDOS CLI Commands
. /
7.19 /
8. EXDOS Error Handling
8.1 User Error Vector
8.2 Disk Errors and Responses
8.3 Error Codes

1. INTRODUCTION

The main code of EXDOS is the FILING SYSTEM HANDLER (FISH) which provides all the facilities necessary for communicating with files on disk. User programs (apart from IS_DOS) do not communicate directly with this, but go through the EXDOS CLI or the DISK DEVICE which in turn calls FISH.

Since FISH is the central part of EXDOS through which all communication goes, this will be described in detail before covering the higher level functions such as the EXDOS CLI and the DISK DEVICE.

1.1 EXDOS ROM ENTRY POINTS

The EXDOS ROM is formatted as an EXOS extension ROM with two additional entry points to FISH and to DISKIO. The DISKIO entry point is provided for user's who wish to use the disk controller hardware directly without going through EXDOS. The FISH entry point is provided to enable IS-DOS to call FISH, and is not intended for user programs. There is of course the normal EXOS system extension entry point which provides various facilities including initialisation and the EXDOS CLI. The format of the start of the EXDOS ROM is as follows:

```
00h     DB      "EXOS_ROM"
08h     DW      0
0Ah     JP      SYS_EXT
0Dh     JP      DISKIO
10h     JP      FISH
```

2. INITIALISATION

During EXOS's cold start routine it will find the EXDOS ROM and include it in its list of extension ROMs. The main stages in EXDOS initialisation which then follow are as below:

1. EXOS calls the EXDOS ROM with action code 7 to ask it how much RAM it requires. EXDOS will ask for sufficient RAM in the system segment for all its internal variables and data areas, including those needed by UNITH, DISKIO and RAMUNIT. This includes three default sector buffers (512 bytes each) so the amount of RAM required here will be about 2.5...3k.

2. Next EXOS calls the EXDOS ROM with action code 0 to initialise it. EXDOS uses this to set up its internal data structures and to initialise its variables. UNITH will be linked in as units 1...4 and any extension unit handlers in other ROMs will be located, linked in and initialised (see PER-3). At this stage EXDOS does not access any disks.

3. Finally EXOS will call each ROM in turn with action code 1 (cold start) to give it the option of starting up. When EXDOS gets this action code it looks on each of its drives in turn for a text file called "EXDOS.INI". If it is found then this is made the default drive and the commands in it executed. These commands may include a "LOAD" command to load and start up an applications program (such as IS-DOS) from disk. If this is not done then EXDOS returns without taking control to give another ROM (for example IS-BASIC) a chance.

2.1 INITIALISATION FILE - "EXDOS.INI"

The file "EXDOS.INI" is a text file containing a sequence of system extension commands which will be executed once when EXOS does its "cold start" ROM scan. The commands can be any arbitrary text strings, which will each in turn be passed around all ROMs, and in particular will be received by the EXDOS ROM itself. Thus any of the EXDOS CLI commands described in section 7 may be included in this file. Also any other extension commands recognised by other ROMs may be included.

For example, it is possible to include a "RAMUNIT" command to set up a RAM disk, followed by "COPY" commands to copy files from disk onto it. After executing these commands, the ROM scan will continue so if the IS-BASIC cartridge is plugged in it will start up. Thus it is possible to use the "EXDOS.INI" file to set up a certain disk environment for other applications ROMs.

Another particularly useful command to include in the "EXDOS.INI" file is "LOAD" to load system extensions or an applications program from disk. Also the "SET" command can be useful to set up EXOS variables. A particular example of an applications program which can be loaded is "ISDOS.COM", which will boot up the IS-DOS operating system. This is described in document PER-16.

When EXDOS first finds the "EXDOS.INI" file, and before executing any commands, it opens an editor channel with associated video and keyboard channels and sets the editor channel up as the default. This allows it to echo the commands to the screen (unless there is a "SET ECHO OFF") and also ensures that the commands have a default channel to read and write to. These channels are closed when all the commands have been executed.

2.2 RE-INITIALISATION

The above procedure is only ever executed when a complete EXOS cold reset occurs which is normally only at power on. When a warm reset occurs for any reason, EXDOS will be re-initialised by EXOS. When this occurs it checks to see if the disk device is still linked in and if not it re-links it in, also it opens any permanent buffer channels (see section 5.2) and generally sorts itself out. It will not re-initialise default settings (such as default drive).

If a load of a new applications program fails then EXOS will do an extension scan with action code 1, which otherwise is only done once at power on. When EXDOS receives this action code it will realise that it is a re-initialisation and will not re-read the "EXDOS.INI" file, but it will automatically re-boot IS-DOS if it was IS-DOS which tried to load the program.

3. THE EXDOS FILING SYSTEM HANDLER - FISH

3.1 FISH FUNCTION CALLS - GENERAL

FISH is called at an entry point at offset 0010h in the EXDOS ROM (see section 1.1). It must always be called in page-3, with the system segment in page-2, the page zero segment in page-0 and anything in page-1. The current stack must be in the system segment or the page zero segment.

All read and write calls are passed the address of a disk transfer area which can be up to 64k in size. The segment or segments containing this area area need not be paged in when FISH is called, but the four EXOS variables FSH_P0....FSH_P3 must be set up to define which four segments correspond to the range 0000h....FFFFh for the disk transfer address. FISH will use these segment numbers in order to access the disk transfer area. It is only strictly necessary to have the ones which actually correspond to the disk transfer area set up, but for safety it is recommended that they are all set up.

All other data areas which are passed to FISH, principally File Control Blocks (FCBs) and pathname strings, must be paged into Z-80 space when FISH is called. Typically they will have to be in page-1 because FISH requires specific segments to be in the other three segments. These data areas can cross segment boundaries provided that BOTH segments are paged in correctly.

Register IY must be set to point to the EXDOS RAM area in page-2, the address of which can be obtained by a system extension command (see section 7.18) and is the same as that required by DISKIO (see PER-2). This address never changes after EXDOS has started up so it need only be obtained once.

Register A is used to pass the function code, other main registers are used for various parameters the details of which vary for different function calls. For many functions IX is a disk transfer address and DE points to an FCB (file control block) which must not cross a segment boundary. HL and BC are also used for some functions.

On return from a FISH function call, the paging and register IY will be preserved. All other main registers (AF, BC, DE, HL, IX and IY) will be corrupted except where they return results from a function call. The alternate register set (AF', BC', DE', HL') will be preserved. All function calls return an EXOS error code in register A, which will be zero if the call was succesful. The flags will be set appropriatly for the value of A.

Below is a complete list of FISH function calls including their function codes. Note that although these function calls are in some cases similar to IS-DOS function calls they are not identical and the function codes are not the same.

Code	Function
00h	Flush buffers
01h	Open file
02h	Close file
03h	Search for first
04h	Search for next
05h	Open searched FCB
06h	Delete file
07h	Create file
08h	Rename file
09h	Move file
0Ah	Change file attributes
0Bh	Get login vector
0Ch	Get allocation information
0Dh	Get or set UPB (unit parameter block)
0Eh	Get file size

05h	Read from file
10h	Write to file
11h	Write with zero fill
12h	Absolute sector read
13h	Absolute sector write
14h	Change directory
15h	Make directory
16h	Remove directory
17h	Get current directory
18h	Set Error vector

3.2 EXOS VARIABLES

EXDOS implements some additional EXOS variables which are used by the various parts of the system. These are listed here. In addition to being accessed as EXOS variables, they can be accessed at address (IY+<name>) as long as IY contains the EXDOS workspace pointer and the system segment is in page-2.

Number	Name	Function
??h	ROM_EXDOS	Segment number of the EXDOS ROM
??h	FSH_P0	\
??h	FSH_P1	\ The four disk transfer area
??h	FSH_P2	/ segments for read and write.
??h	FSH_P3	/
??h	ECHO	EXEC flag for batch files
??h	VERIFY	EXDOS verify flag (ON/OFF)
??h	DEF_UNIT	Default unit (drive) number (1...26)
??h	RAM_UNIT	RAM disk unit number. Zero if none
??h	STEP_RATE	Step rate for UNITH (0...3)
??h	DISK_ERR	Error code for .ABORT
??h	RND_0	\
??h	RND_1	\ 32 bit random number maintained
??h	RND_2	/ by DISKIO for volume ids.
??h	RND_3	/

3.3 FILE CONTROL BLOCKS (FCBs)

All file access is controlled by FCBs which are similar, but not identical to extended FCBs in MS-DOS, which are in turn an extended version of CP/M FCBs. Whenever an FCB is passed to FISH, it must be paged into Z-80 address space. This means that IS-DOS has to copy all user FCBs to an internal buffer before passing them on to FISH, which enables it to also make the translation from CP/M to FISH format and back again. Details of formats of IS-DOS FCBs can be found in document PER-16.

An unopened FCB only has the drive, filename and extension fields set up, and these may be ambiguous (containing "?" characters). When an unopened FCB is passed to FISH, a PATHNAME string is also passed, which defines what directory or sub-directory to use. See section 3.3.2 for details of these strings.

When an OPEN or CREATE operation is done on an FCB, the remaining fields are filled in. The reserved area contains information which specifies where the file is located on disk and what directory it is in, so the pathname string is no longer required.

3.3.1 FCB FORMAT

All FCBs (File Control Blocks) passed to FISH are 48 bytes long, although some of the fields at the end will not be used for unopened FCBs.

00h	0FFh	(For compatibility with MS-DOS)
01h...05h	000h	(.. .. ..)
06h	File attribute. Zero for normal file, can be used to search for special directory entries (see section 3.7) and to create hidden files.	
	b0 - Read only	
	b1 - Hidden file	
	b2 - System file	
	b3 - Volume label	
	b4 - Sub-directory	
	b5 - Archive	
	b6,b7 - Zero	
07h	Drive number.	00h => default drive
		01h...1Fh => specified drive
		20h...FFh not allowed
08h...0Fh	Filename left justified with trailing blanks, bit-7 will be ignored.	
10h...12h	Filename extension.	
13h...14h	Not used (Block number in IS-DOS, extent number and S1 in CP/M)	
15h...16h	Not used (Record size in IS-DOS, S2 and record count in CP/M)	
17h...1Ah	File size in bytes, lowest byte first.	
18h...1Ch	Date of last modification. bits 15...9 = year 0...119 (1980...2099) bits 8...5 = month 1...12 bits 4...0 = day 1...31	

Pathnames are used by all FISH calls which take unopened FCBs as parameters. These are:

Open file
Create file
Delete file
Rename file
Search for first
Change file attributes
Get file size
Move file

3.4 FUNCTION CALL 00h - FLUSH BUFFERS

Parameters: None
Results: A = 0 (status code)

This function flushes all dirty sector buffers to the disk. Open files are not closed. It takes no parameters and always returns a zero status code.

3.5 FUNCTION CALL 01h - OPEN FILE

Parameters: DE -> Unopened FCB
HL -> Pathname string
Results: A = Status code
DE -> Same FCB with some fields filled

The specified directory is is searched for the file specified in the FCB and the file is opened, an error (.NOFIL) is returned if it is not found. If drive specified is zero (default drive) then the drive number in the FCB will be replaced by the actual drive number used (1...26). The ambiguous character "?" is allowed in the filename or extension in the FCB, in which case the first file found will be opened.

The file size, date, time and attribute byte and the current volume id are copied into the FCB and the file pointer is set to zero. The FCB can be used immediately for reading or writing.

1Dh...1Eh	Time of last modification. bits 15..11 = hours 0...23 bits 10...5 = minutes 0...59 bits 6...0 = seconds 0...29 (2 seconds)
1Fh...26h	Reserved.
27h	Not used (Current record in IS-DOS and CP/M)
28h...2Bh	File pointer, lowest byte first. Set to zero on open and adjusted by all reads and writes. (Random record in IS-DOS and CP/M.
2Ch...2Fh	Volume id. This is set up when the file is opened and checked on every access of the FCB to ensure that the correct disk is being used.

3.3.2 PATHNAME STRINGS

Whenever an unopened FCB is passed to FISH, a pathname string is also passed, which must be paged into Z-80 memory space as well as the FCB. This string can specify which drive and directory to use for the operation. If it is a null string then the current directory (root directory by default) of the drive specified in the FCB will be used, with zero being replaced by the current default drive.

If the string is non-null then it contains an optional drive specifier (for example "A:", "z:", "12:"), followed by a series of sub-directory names separated by back-slashes ("\").

If the drive specifier is present then the appropriate drive number will be put in the FCB, over-riding whatever was there before. If no drive specifier is given then the drive number in the FCB will be used, which will normally be set to zero thus selecting the default drive.

If the first character after the ":" is a back-slash then the sub-directory list starts from the root directory, otherwise it starts from the current directory of the drive. Each sub-directory name in turn will be found until the end of the string, with a .NODIR error being returned if any one does not exist. The sub-directory names may not contain ambiguous characters ("?" or "*").

3.6 FUNCTION CALL 02h - CLOSE FILE

```
Parameters:     DE -> Opened FCB
Results:         A =  Status code
                DE -> Same FCB
```

Closes the file corresponding to the FCB. It is not necessary to close files which have only been read and if they are closed the disk directory is not updated. If any data has been written to the file then the directory entry will be updated appropriately with the new date, time, size and allocation information. Note that an altered file can only be closed correctly if the filename in the FCB has not been changed. Also the attribute byte is not altered, this can only be done with a "change attributes" call (see section 3.14).

3.7 FUNCTION CALL 03h - SEARCH FOR FIRST

```
Parameters:     BC -> Space for new FCB (40 bytes)
                DE -> Unopened FCB
                HL -> Pathname string
Results:         A =  Status code
                IX -> Same address
                DE -> Same FCB
```

Searches the specified directory for a file which matches the filename in the FCB. Ambiguous filenames (containing "?" character) are allowed, in which case the first file which matches will be found.

When a file is found, a valid unopened FCB for this file will be setup at the address pointed to by BC. This will consist of the first 7 bytes from the search FCB followed by the drive number and then the 32 byte directory entry. Note that the attributes byte in the FCB (byte 6) will be the same as the search FCB, NOT the attributes of the located file. This files attributes can be found from the directory entry (see section 4.3).

If the search FCB attributes byte is zero then only normal file entries will be found. Entries for volume label, sub-directories, hidden files and system files will not be returned.

If the "volume label" bit of the search attributes is set then only the volume label entry will be returned, and it will always be looked for in the root directory regardless of the pathname. If the "hidden file", "system file" or "sub-directory" bits are set then all normal file entries as well as the specified special entries will be returned. See section 4.3.1 for details of the bit assignments and meanings.

The file can be opened by immediately calling the "open searched FCB" function (see section 3.9). This uses information stored in the search FCB to define which directory the file is in. Alternatively the original pathname string can be passed to an "open file" function (see section 3.5) along with the new FCB but this will be less efficient as the whole sub-directory path will have to be re-searched rather than just the last directory.

3.8 FUNCTION CALL 04h - SEARCH FOR NEXT

```
Parameters:     BC -> Space for new FCB (40 bytes)
                DE -> FCB used for "search for first"
Results:         A =  Status code
                IX -> same address
                DE -> same FCB
```

After a "search for first" function call (see section 3.7), this function may be called repeatedly to find subsequent matches of the filename if it was ambiguous. The address pointed to by BC can be different if desired but must be paged into Z-80 memory space of course.

The reserved section of the search FCB contains the information which FISH needs to continue the search, so no file operations must be performed with this FCB between the "search for first" and any calls to this function. The information stored in the new FCB is exactly as for "search for first".

3.9 FUNCTION CALL 05h - OPEN SEARCHED FCB

```
Parameters:     DE -> FCB returned by search function
                HL -> Search FCB.
Results:         A =  Status code
                DE -> Same FCB opened
```

This function must only be called to open a file which has been found by "search for first" or "search for next" (see section 3.7). It uses information stored in the search FCB to locate the directory containing the file without having to rescan down a chain of sub-directories and it thus more efficient than doing an "open file".

The FCB pointed to by DE should be the one set up by the search operation and it will be opened just like a normal open function call. The search FCB must be the FCB which the search operation used and must not have been altered since this file was found.

3.10 FUNCTION CALL 06h - DELETE FILE

Parameters: DE -> Unopened FCB
HL -> Pathname string
Results: A = Status code
DE -> Same FCB

All entries in the specified directory which match the filename in the FCB are deleted. The "hidden file" and "system file" bits in the attributes byte can be set to enable these type of files to be deleted, other bits must be clear. See section 4.5 on support for un-deleting files. If no files are deleted then a .NOFIL error will be returned.

3.11 FUNCTION CALL 07h - CREATE FILE

Parameters: DE -> Unopened FCB
HL -> Pathname string
Results: A = Status code
DE -> Same FCB opened

This function is very similar to the "open file" function call (see section 3.5) except that it is used to create a new file rather than open an existing one. Ambiguous filename characters ("?") are not allowed. If the file already exists in the specified directory then it will be truncated to zero length ready for writing, otherwise the first free directory entry will be set up for this file, initially with zero length. If the directory is full then a .DRFUL error will be given (only the root directory can ever be full).

The "hidden file" bit may be set in the attribute byte to exclude the file from normal directory searches. All other bits in the attributes byte will be ignored.

3.12 FUNCTION CALL 08h - RENAME FILE

Parameters: DE -> Unopened modified FCB
HL -> Pathname string
Results: A = Status code
DE -> Same FCB

The modified FCB has a second filename starting at byte 18h in the FCB. Ambiguous characters ("?") are allowed in either filename.

The specified directory will be searched and every matching occurence of the first filename will be replaced with the second. If a "?" appears in the second name then the corresponding character from the original name will be preserved. Checking is done to ensure that no duplicate filenames are created. If no files are renamed then a .NOFIL error will be returned.

The attributes byte allows the volume name, sub-directories, hidden, system and read only files to be renamed by setting the appropriate bits. If the "volume name" bit is set then ONLY the volume name will be renamed and the root directory will be used regardless of the pathname.

If sub-directories are being renamed then a check will be made with the current directory path and any directory which is contained in this path will not be renamed.

3.13 FUNCTION CALL 09h - MOVE FILE

```
Parameters:    DE -> Search FCB
               HL -> New pathname string
Results:        A =  Status code
               DE -> Same FCB unmodified
```

This function must only be called after a file or sub-directory has been located with a "search for first" or "search for next" function call (see section 3.7). It is used to move the file or sub-directory from one directory to another. The data of the files is not duplicated. It is not possible to move a sub-directory into one of its own decendent directories or into itself. Also any directory which is contained in the current directory path cannot be moved.

The search FCB contains information which defines where the current directory entry for the file is, and the new pathname string defines which directory the file is to be moved to. If the new pathname string contains a drive specifier then it must be the same drive as the file is currently on. The whole directory entry corresponding to the file will be copied into the new directory and deleted from the old one. The search FCB is not modified at all so more "search for next" function calls can be made with it.

3.14 FUNCTION CALL 0Ah - CHANGE FILE ATTRIBUTES

```
Parameters:    DE -> Unopened FCB
               HL -> Pathname string
                B =  0 => read attributes
                     1 => change attributes
                C =  New attribute byte if B=1
Results:        A =  Status code
               DE -> Same FCB
                C =  Attribute byte
```

If B=0 then the attribute byte of the specified file or sub-directory will be returned (see section 4.3.1 for details of the attribute byte). If B=1 then the attributes will be set to the specified new value and the new value returned, but only the "system file", "hidden file", "read only", and "archive" bits can be changed. Ambiguous filenames are not allowed.

Note that in order to change the attributes of hidden files, system files or sub-directories, the appropriate attribute bits must be set in order to find the file in the first place.

3.15 FUNCTION CALL 0Bh - GET LOGIN VECTOR

```
Parameters:    None
Results:        A =  0 (status code)
               DE =  High word of login vector
               HL =  Low word of login vector
```

The login vector is a 26 bit number which specifies which logical drives are available to be used. Unlike CP/M all drives in the system are always "logged in". Bit-0 corresponds to drive "A:" and so on, with the bit being set to indicate an available drive.

3.16 FUNCTION CALL 0Ch - GET ALLOCATION INFORMATION

```
Parameters:    ??????????
Results:       ??????????
```

Yet to be defined in detail. Will return at least:

sectors/cluster
sector size (always 512)
number of clusters on disk
number of free clusters

3.17 FUNCTION CALL 0Dh - GET OR SET UPB

Parameters: DE -> Pointer to UPB (or 64 byte space for one).
C = Drive number (0...26)
B = 0 => Get UPB
1 => Set UPB
2 => Cancel UPB

Results: A = Status code
DE -> Same address.

If B=0 then the current UPB (unit parameter block) for the specified drive will be copied into the user's memory address. The UPB will always be checked with the disk when this function is called to ensure that it is up to date. The format of UPBs is described in document PER-3.

If B=1 then the user's memory area must contain a UPB which will then be used for all further accesses to this drive, rather than determining the UPB from the disk. Any dirty buffers will be flushed to disk first. This function is provided to allow some non-standard disk formats to be used.

A call with B=2 will cancel the effect of the user specified UPB, allowing the drive to return to normal automatic disk parameter selection. In this case, register DE need not be set up. When this function is called any buffers will be flushed and the UPB cancelled. The NEXT access to this drive will then determine the disk type from the disk. The disk should therefore not be changed until after this function is issued to ensure that buffers are flushed correctly.

3.18 FUNCTION CALL 0Eh - GET FILE SIZE

Parameters: DE -> Unopened FCB
HL -> Pathname string
Results: A = Status code
DE -> Same FCB with size, date and time

The directory is searched for the specified file (ambiguous characters "?" are not allowed) and the size, date, and time fields from the directory entry are copied into the FCB.

3.19 FUNCTION CALL 0Fh - READ FROM FILE

```
Parameters:     IX -> Disk transfer address
                DE -> Opened FCB
                BC =  Number of bytes to read
Results:         A =  Status code
                IX -> Updated memory address
                DE -> Same FCB
                BC =  Number of bytes read
```

Reads the specified number of bytes from the file to the disk transfer address. Up to 64k can be read into the segments defined by FSH_P0...FSH_P3 with any segment boundaries being handled automatically, an error will be returned if the read attempts to wrap around past the top of page-3.

The position in the file is determined by the file pointer field of the FCB which will be updated appropriately. If an error code is returned then the number of bytes succesfully read may be less than the number requested. If the read attempts to read beyond the end of file then a .EOF error will be returned.

3.20 FUNCTION CALL 10h - WRITE TO FILE

```
Parameters:     IX -> Disk transfer address
                DE -> Opened FCB
                BC =  Number of bytes to write
Results:         A =  Status code
                IX -> Updated memory address
                DE -> Same FCB
                BC =  Number of bytes written
```

Writes the specified number of bytes from the disk transfer address to the file. Up to 64k may be written from the segments defined by FSH_P0...FSH_P3, with any segment boundaries being handled automatically. An error will be returned if it attempts to write over the end of page-3. If the EXOS variable "VERIFY" is zero then an automatic verify of the written data will be performed.

The position in the file is determined by the file pointer field of the FCB which will be updated appropriately. If an error code is returned then the number of bytes succesfully written may be less than the number requested.

If the write goes beyond the end of file then additional disk space will be allocated as necessary and the file size adjusted. If it starts beyond the end of file then disk space to fill the "gap" will be allocated but not initialised.

If there is insufficient space on the disk for the entire write operation, then a .DSKFL error will be returned and NO DATA WILL BE WRITTEN even if the start of the write would overwrite existing data in the file.

3.21 FUNCTION CALL 11h - WRITE WITH ZERO FILL

Parameters:
DX -> Disk transfer address
DE -> Opened FCB
CX = Number of bytes to write

Results:
AL = Status code
IX -> Updated memory address
DE -> Same FCB
BC = Number of bytes written

This function is identical with "write to file" (above) except that if the write starts beyond the end of the file, the disk space allocated to fill the "gap" will be initialised to zero.

3.22 FUNCTION CALL 12h - ABSOLUTE SECTOR READ

Parameters:
IX -> Disk transfer address
DE = Logical sector number
AH = Number of sectors to read
AL = Drive number (0...26)

Results:
AL = Status code
IX -> Updated transfer address
DE = Next logical sector number
AH = Number of sectors read
AL = Same drive number

The specified sectors are read from the disk to the disk transfer address. Up to 64k can be read into the segments defines by FSH_P0...FSH_P3, with segment boundaries being handled automatically and an error will be returned if the read attempts to read beyond the end of page-3. If an error occurs then the number of sectors read may be less than the number requested. Normal retries are provided in case of errors.

3.23 FUNCTION CALL 13h - ABSOLUTE SECTOR WRITE

```
Parameters:     IX -> Disk transfer address
                DE =  Logical sector number
                 H =  Number of sectors to write
                 L =  Drive number (0...26)
Results:         A =  Status code
                IX -> Updated transfer address
                DE =  Next logical sector number
                 H =  Number of sectors write
                 L =  Same drive number
```

The specified sectors are written from the disk transfer address to the disk. Up to 64k can be written from the segments defined by FSH_P0...FSH_P3, with segment boundaries being handled automatically and an error will be returned if the write attempts to write beyond the end of page-3. If an error occurs then the number of sectors written may be less than the number requested. Normal retries are provided in case of errors. If the EXOS variable "VERIFY" is zero then an automatic verify of the sectors will be done.

3.24 FUNCTION CALL 14h - CHANGE DIRECTORY

```
Parameters:     HL -> Pathname string
Results:         A =  Status code
```

The pathname specifies a path to a directory which is to be made the current directory for this drive. If any directory in the pathname does not exist then a .NODIR error is returned and the current directory is not changed.

3.25 FUNCTION CALL 15h - MAKE DIRECTORY

```
Parameters:     HL -> Pathname string
Results:         A =  Status code
```

The pathname specifies a new sub-directory which is to be created. The last item in the string is the name of the new sub-directory to create. If it already exists then a .DIRX error is returned otherwise the new sub-directory is created and the two initial entries in it set to "." which points to the directory itself, and ".." which points to its parent directory.

3.26 FUNCTION CALL 16h - REMOVE DIRECTORY

```
Parameters:     HL -> Pathname string
Results:         A =  Status code
```

The pathname specifies a sub-directory which is to be removed. If the specified directory is not empty (apart from the "." and ".." entries) then a .DIRNE error is returned, otherwise the sub-directory is removed from its parent directory. The current directory for a drive cannot be removed and neither can the root directory.

3.27 FUNCTION CALL 17h - GET CURRENT DIRECTORY

```
Parameters:     HL -> Space for string (64 bytes)
                 B =  Drive number (0...26)
Results:         A =  Status code
                HL -> Pathname string at same address
```

The full pathname (starting from the root directory) of the current directory for the specified drive is copied into the user's data area. It will not include the drive indicator or an initial "\". If the root directory is the current directory then a null string will be returned.

3.28 FUNCTION CALL 18h - SET ERROR VECTOR

```
Parameters:     EL = Address of new vector
                 B = Segment of new vector
Returns:         A = 0 (status code)
                EL = Address of old vector
                 B = Segment of old vector
```

The error vector allows the user program to intercept disk errors and thus to override the normal prompts. Section 6 describes the facility in detail. This function allows the user to specify the address of a routine which will be called before EXDOS's error handling. The user's routine will always be entered in page-3.

The address of the previous routine is returned so that if these results are saved and then passed back to this function call later, the old routine will be restored. If the segment number is zero, as it is by default, then the routine will not be called.

4. FILING SYSTEM and DISK ALLOCATION

The filing system supported by EXDOS is compatible with all versions of MS-DOS up to version 2.00 (at least), and with MSX-DOS, provided 512 byte sectors are used. The various disk formats are handled by the unit handler (see PER-3 and PER-4), EXDOS simply sees a disk divided up into 512 byte sectors numbered 0...N-1 where "N" is the total number of sectors on the disk.

Each disk has a boot sector (sector 0), a root directory and one or more copies (usually 2) of a file allocation table (FAT). The remainder of the disk is divided up logically into CLUSTERS numbered 2....M+1 where M is the total number of clusters. A cluster is the unit of allocation on the disk and the size of a cluster depends on the disk format, normally it is one or two sectors. The sectors of a cluster are always sequential.

The root directory contains a fixed number (usually 64 or 112 depending on the disk format) of directory entries, each of 32 bytes. Each directory entry refers to a file, sub-directory or the volume label. A sub-directory also consists of a series of 32 byte directory entries, but is a variable size, with more space being allocated when it becomes full. The format of directory entries is exactly as for MS-DOS 2.00 (see section 4.3).

A file or a sub-directory consists of a chain of clusters, with the first cluster number being stored in the directory entry for the file. The remaining clusters of the file can be found by following a chain of cluster numbers in the FAT. The FAT also indicates which clusters on the disk are free.

4.1 BOOT SECTOR

The boot sector is always the first sector of the disk, put there by the FORMAT and DISKCOPY programs. This format is designed to be compatible with all versions of MS-DOS up to version 2.00 (at least) and with MSX-DOS. Although it is called the boot sector it does not contain any boot code used by EXDOS since this facility is provided by the "EXDOS.INI" file (see section 2.1).

```
00h - 0EBh  \  Dummy 8086 jump instruction
01h - 0FEh   >  for MS-DOS compatibility.
02h - 090h  /

03h...0Ah - "EXDOS1.0"  System identification string.
```


0Bh...1Dh - DPB (may be absent on early MS-DOS disks). See "unit handler specification" (PER-3) for details.

1Eh - 2-80 "RET" instruction to ensure MSX compatibility. MSX uses this as the entry point to a boot program which boots up MSX-DOS if present. Not used by EXDOS which uses the "EXDOS.INI" facility for starting IS-DOS.

1Fh...3Fh - Not used. This is 33 bytes reserved for compatibility with possible future extensions to the MS-DOS or MSX-DOS boot sector definition.

40h...45h - "VOL_ID" String to indicate that volume ID and dirty disk flag are present.

46h - Dirty disk flag (see section 4.5).
0 => clean disk
1 => dirty disk

47h...4Ah - 4 byte unique volume ID. This is a random number put on by FORMAT and DISKCOPY to allow full checking of disk changes (see section 4.4).

4Bh..1FFh - Not used

4.2 FILE ALLOCATION TABLE

The file allocation table consists of a 12-bit entry for each cluster on the disk. The first two entries correspond to clusers 0 and 1 which do not exist, so the FAT entry for the first cluster (cluster 2) starts in the fourth byte of the FAT. The first three bytes in the FAT are as follows:

Byte 0: FATID byte, always F8h...FFh. (see PER-3)
Byte 1: Always FFh
Byte 2: Always FFh

The FATID byte is used to determine disk format if there is no DPB in the boot sector (see PER-3), otherwise it is not used.

Each cluster entry is zero if the cluster is free otherwise the cluster is part of a file and the value of the entry is the number of the next cluster in the file, with FF8h...FFFh indicating the last cluster in the file. Thus the clusters which make up the file are chained together in the file. The values FF0h...FF7h are used to indicate reserved clusters if they are not part of a chain, with FF7h indicating a bad cluster which must not be used.

A disk can contain any number of copies (normally two) of the FAT. The muitiple copies are used in case the first copy becomes unreadable due to a disk error. Normally all copies are identical but see section 4.5 for an exception to this (to allow un-deletion).

4.3 DIRECTORY ENTRY FORMAT

All directory entries in the root directory or sub-directories are 32 bytes long with the following format:

00h...07h - Filename. 8 characters, left justified and padded with spaces. The first byte of this indicates the status of the entry:

00h - Never been used.
E5h - Entry deleted.
05h - First filename character is E5h

Any other character is the first character of the filename. Note that the two special names "." and ".." are reserved to point to this directory and its parent.

08h...0Ah - Filename extension.

0Bh - Attribute byte. See section 4.3.1.

0Ch...15h - Reserved.

16h...17h - Time \ Format exactly as in FCB
18h...19h - Date / (see section 3.3.1)

1Ah...1Bh - Starting cluster of file (0002h...0FFF7h).

1Ch...1Fh - Size of file in bytes, lowest byte first.

4.3.1 FILE ATTRIBUTE BYTE

The allocation of bits in the file attribute byte in the directory entry is given below. The same bit assignments are used for the attributes byte in an FCB (see section 3.3.1). Note that certain combinations of bits are meaningless (both volume label and sub-directory set for example). See section 3.7 for how to use these attributes in directory searches, and section 3.14 for how to change them.

Bit 0 - Read only. If set then this file cannot be written to or deleted, but can be read or renamed.

Bit 1 - Hidden file. If set then this file will be excluded from normal directory searches and thus from directory listings.

Bit 2 - System file. If set then this file will be excluded from normal directory searches.

Bit 3 - Volume label. If set then this entry defined the name of this volume. Can only occur in the root directory.

Bit 4 - Sub-directory. If set then this entry is a sub-directory rather than a file and so cannot be opened for reading or writing.

Bit 5 - Archive bit. This bit is set whenever the file has been written to and closed. It can be examined and reset by an ARCHIVE program to determine whether this file has been changed and should be backed up.

Bit 6 - Reserved (always 0).

Bit 7 - Reserved (always 0).

4.4 VOLUME ID

The format of the boot sector allows an 4 byte unique volume identification to be stored (see section 4.1). This is a random number put there by the FORMAT and DISKCOPY programs to distinguish this disk from all others. There is also a "VOL_ID" string which can be checked to determine whether the volume id is present. This ensures compatibility with MS-DOS and MSX-DOS.

When a unit handler (such as UNITH) gets a BUILD BPB command it returns the volume id from the disk (if it is present) to EXDOS. EXDOS uses this function, in conjunction with MEDIA CHECK, to determine whether the disk has been changed for a different one. If so, and if EXDOS has any dirty buffers, then the user will be warned (see section 8.2). This ensures that dirty buffers can never be flushed onto the wrong disk.

When a file is opened, or a "search for first" function call done, EXDOS copies the current volume id for the drive into the FCB. All future uses of this FCB then check that the volume id is correct. If not the user is given the option of inserting the correct disk, or using the current one anyway (see section 8.2). This ensures that data can never be read from or written to the wrong disk by accident.

4.5 DIRTY DISK FLAG (UN-DELETION)

Normally in MS-DOS compatible systems, when a file is deleted the clusters which were allocated to it are marked as free in all copies of the FAT. This means that there is no way to un-delete a file which has been accidentaly deleted since the chain cannot be re-built. EXDOS however allows files to be un-deleted as long as no space allocation operations have been performed since the delete (ie no files created of data written to disk). This facility is only available on disks which contain the "volume id" in the boot sector. Thus if a disk was created by MS-DOS or MSX-DOS then files cannot be un-deleted from it.

In EXDOS when a file is deleted from a disk with a valid volume id, the clusters are freed in all copies of the FAT except the last one, and the "dirty disk" flag in the boot sector is set to 1. Thus the cluster chain for the file is still available in the last copy of the FAT. The CHKDSK program, which scans the FATs and directories, can therefore re-build the deleted file (or files).

When any cluster allocation operation is performed, the dirty disk flag is examined. If it is zero then the operation continues normally. However if it is non-zero then the last copy of the FAT will be updated to be the same as the others before looking for a free cluster. This is necessary to ensure that the last copy of the FAT does not contain a mixture of the remnants of files which have been deleted and then the space re-allocated.

5. DISK BUFFERING

Where possible, EXDOS does data transfers directly into user's memory. However when incomplete sectors are transferred, or transfers run across segment boundaries, it is necessary for buffering to be used. Buffers are also needed for EXDOS to manipulate directories and FATs. EXDOS therefore maintains a linked list of 512 byte sector buffers.

EXDOS permanently allocates itself three sector buffers when it is initialised. These are in its system segment RAM and can never be removed. Additional sector buffers can be created at any time by open a channel to a special device called "BUFFER:" which EXDOS links in.

The order of buffers on the chain is continucusly being changed by EXDOS to provide a priority scheme so that frequently accessed data, such as FAT sectors tend to stay resident in sector buffers when they are needed.

5.1 CREATING ADDITIONAL SECTOR BUFFERS

To create some extra disk buffers, the user simply opens a normal EXOS channel to device "BUFFER:", using the unit number to specify how many buffers he wants in the range 1...10. There can be as many separate buffer channels open as desired, although if too many buffers are added then the system can slow down as it takes EXOS longer to search its buffer chain.

When the buffer channel is opened, the disk device will obtain the required amount of channel RAM (about 540 bytes per buffer), set up the sector buffers and link them into EXDOS's buffer chain. EXDOS will then use the new buffers freely until the channel is closed. When this occurs, any data in the buffers is flushed and the buffers are unlinked from the chain. If the channel RAM for a buffer channel is moved then the pointer which make up the buffer chain will be adjusted to keep the chain intact.

If a system reset occurs then a "forced close" of the buffer channels will occur without calling the buffer device, and the EXDOS ROM will be re-initialised (along with all other ROMs). When this occurs EXDOS will lock along its chain of buffers (which will still be intact since EXOS does not zero the RAM) and flush any that are dirty. It then removes any extra buffers from its chain so it is back to using the default three buffers until a new buffer channel is open.

5.2 ADDITIONAL PERMANENT BUFFERS

The EXDOS CLI "BUFFERS" command can be used to specify a number of permanent buffers which EXDOS will maintain in addition to its three default buffers. EXDOS uses channel number 254 to provide these buffers. When the "BUFFERS" command is received (typically in the EXDOS.INI file), EXDOS opens an appropriate buffer channel using channel number 254. It remembers the number of buffers requested and after any system reset it will re-open the channel. If the channel is closed by the user it will note this fact and re-open it as soon as it gets a chance (next system extension scan for example).

The number of additional permanent buffers can be changed at any time by giving an appropriate "BUFFERS" command, the old channel will be closed before opening the new channel which may have fewer or more buffers. A "BUFFERS 0" command will remove all additional permanent buffers

6. DISK DEVICE

The disk device driver is linked into EXOS as a "user" device when EXDOS starts up, and re-linked whenever a system reset un-links it. (A system reset is always followed by re-initialising system extensions.)

Although the disk device is described as a single device, several device descriptors are in fact linked in for it. There is one device descriptor with the name "DISK", which is set up as the EXOS default device. When this device is used the unit number specifies which drive is required.

In addition a separate device descriptor is linked in for every drive which is supported, with device names "A", "B",.... "Z". When these devices are used, the unit number passed by the user is ignored and the device name determines the drive. This allows the user to refer to the disk drives by letters rather than numbers, to be consistent with other operating systems (CP/M, MSX-DOS and MS-DOS for example).

The unit number given when opening a channel to device "DISK" is interpreted as follows:

0	Current default drive. If the user does not specify a unit number to the "DISK" device then EXOS will use zero.
1...26	Drives A: to Z: (also 1: to 26:)
27..255	Not allowed

6.1 DISK I/O CHANNELS

Disk I/O channels are always opened (or created) to a specific file, with a default filename being used if the user gives a null filename. The OPEN and CREATE functions match up exactly to the appropriate FISH calls described in section 3. The filename given by the user is passed directly to an "open channel" or "create channel" function call after taking the last item off as the filename, and storing it in an FCB which the disk device sets up in its channel RAM area. This allows sub-directories to be accessed by means of a path name.

Data can be read or written using the EXOS block or character function calls which are easily translated into FISH read and write calls (writing is done without zero fill). Random access can be performed by using the EXOS "set and read channel status" function to read the current value of the 32 bit file pointer or to move it. This function also returns the file size. See the EXOS kernel specification for details (protection byte not supported).

The "read status" EXOS function returns either C=0FFh (EOF) if the file pointer is at or beyond the end of file, and C=0 (character ready) otherwise. It never returns C=1.

The only special function call which the disk device supports is FLUSH to ensure that any dirty buffers have been written to disk. This is the same special function code (@@FLSH=16) as used by the NETWORK device. Although this function must be sent to a specific disk channel, it will flush all dirty buffers for any disk channels.

7. EXDOS CLI COMMANDS

The EXDOS CLI interprets system extension command strings which are passed around system extensions by a "scan extensions" EXOS call. This call can be made by any applications program and so the EXDOS CLI commands are available from any program which supports these commands. If it is possible to use the "HELP" facility in a program, then all the EXDOS CLI commands will be available.

Many of the commands use the default channel for printing results and messages, and also for reading the response to user prompts. This default channel will also be used for any EXDOS disk error prompts which occur. Normally the default channel will be the main editor channel. If an error occurs during any of these commands then the EXOS error code will be returned and normally the calling program will the display the appropriate error message

Below is a list of all the internal commands contained in the EXDOS ROM. These are explained in more detail in the sections which follow.

```
DIR      - prints directory of disk
DEL      - deletes a disk file
REN      - renames a disk file
ATTR     - changes attributes of disk file
TYPE     - types a file on the screen
COPY     - copies files
MOVE     - moves files from one directory to another
DATE     - sets or displays system date
TIME     - sets or displays system time
SET      - sets or displays EXOS variables
LOAD     - loads various module formats
ASSIGN   - maps two drive identifiers to one drive
MAPDISK  - Allows one drive to be used for two disks
BUFFERS  - creates "permanent" disk buffers
RAMUNIT  - installs or removes the RAM disk
FORMAT   - formats blank disks
EXDOS    - System use

CD       - change current directory path
MD       - make a new sub-directory
RD       - remove a sub-directory
```

7.1 COMMAND PARAMETERS

EXDOS CLI commands such as "COPY" take parameters specifying the source or destination of data. There are various different ways in which this can be specified, not all of which are allowed in all commands. The allowed types for each command are listed with the command descriptions below. This section specifies the various types of parameter:

<File> Example: A: 2:FRED.COM x\y123\XYZ.BAT
This can be any string which is acceptable to an "open channel" FISH function, using an indirect FCB. It may specify the drive, a directory path and a filename, all of which are optional although some commands may insist on the filename.

<chan> Example: £0 £107 £222
This is a channel number which specifies an EXOS channel which must be already open. Data will be read from or written to this channel as appropriate for the command.

<dev> Example: NET0: TAPE:FRED DISK3:X\Y\Z
This is a string which will be given to EXOS to open or create a channel with. It is used in some commands if FISH rejects the string parameter with a .NDISK error. EXDOS will open or create the channel, read or write data from it and then close it. It uses channel numbers £253 and £252 for this.

7.2 DIR COMMAND

DIR <file>

The DIR command prints a directory listing to the default channel. The parameter "<file>" can be any valid pathname with or without a filename, or can be omitted altogether. If a filename, which may be ambiguous, is included then only files which match the filename will be listed, otherwise all files in the specified directory will be listed. If the pathname is ommitted then all files in the current directory for that drive will be listed.

The directory listing includes the volume name (if present), the full path of the directory being listed and the amount of free space on the disk. For each file the listing gives its name, size in bytes, date and time of last modification and an indication of whether it is read only. Sub-directories are included in the directory listing. Hidden files and system files are not listed.

7.3 DEL COMMAND

DEL <file>

This command deletes all files in the specified directory which match the filename. If the filename is omitted or is "*.*" then all files in the directory will be deleted but EXDOS will first prompt:

Are you sure (Y/N)?

Sub-directories cannot be deleted with this command, and neither can system files or read only files.

7.4 REN COMMAND

REN <file> <file>

The first "<file>" specifies a directory path and filename. The second "<file>" is just a filename with no directory path or drive name. Both filenames can be ambiguous, and the "filenames" can in fact specify sub-directories allowing these to be renamed, although any directory in the current directory path for a drive cannot be renamed.

All files in the directory which match the first filename will be changed to the second. Character positions corresponding to "?" or "*" in the second filename will remain unchanged. The command will not create duplicate files in the directory. An attempt to do this will result in a .FILX error. If the new filename would have embedded spaces in then they will be replaced with "_".

7.5 ATTR COMMAND

Format yet to be defined. This command allow the read only and hidden bits of a file or sub-directory's attributes to be changed.

7.6 TYPE COMMAND

TYPE <file>|<dev>|<chan>

Data will be read from the specified source and written to the default channel using EXOS single character reads and writes. TAB characters will be expanded to eight character boundaries and other control characters will be suppressed (apart from CR and LF). The process will continue until an error occurs on the source (such as end of file) or a Ctrl-Z character is read.

7.7 COPY COMMAND

COPY <source> <dest>

This command copies data from one file to another. More than one file can be copied by specifying an ambiguous filename and the data can be copied to any other device or channel rather than a disk file. The copy operation will be done using large block reads and writes to minimise selection of disk drives.

7.8 MOVE COMMAND

MOVE <file1> <file2>

The "<file2>" parameter must specify a directory path without a filename and must be unambiguous. The "<file1>" parameter is a filename which can be ambiguous and can include a directory path. All files and sub-directories which match <file1> will be moved to the directory specified by <file2>. The data of the files is not copied, only the directory entries are manipulated. A sub-directory cannot be moved if it is in the current directory pathname, and also a sub-directory cannot be moved into one of its own decendents.

7.9 DATE COMMAND

DATE string

The system date will be set to value specified in the string using the format is "DD-MM-YY". Any reasonable separator can be used in place of the "-" and if the month and or year is omitted then it will be left unchanged. If there is no string parameter at all then the current date will be displayed.

7.10 TIME COMMAND

TIME string

The system time will be set to the value specified in the string using the format "HH-MM-SS" (24 hour clock). Like DATE, any reasonable separator can be used in place of the "-" and the minutes and seconds can be ommitted if desired. If the string is omitted altogether then the current time will be displayed.

7.11 SET COMMAND

SET <variable> <value>

Sets EXOS variables. The <variable> and <value> parameters each specify an 8 bit number. If space allows then string parameters will be supported with "ON" and "OFF" in addition to number for the value and various keys for the variable names.

7.12 LOAD COMMAND

LOAD <source>

The "source" parameter can be a channel number, device or disk file. If it is a disk file or device then an EXOS channel will be opened and will be used for an EXOS "load module" function call. This allows system extensions (such as new unit handlers) or new applications programs (such as IS-DOS) to be loaded. EXOS does not support any module header types itself.

7.13 ASSIGN COMMAND

ASSIGN <drive1>=<drive2>

This command sets <drive1> up as another name for <drive2>. It is implemented at a very low level so any file open to <drive1> will get "redirected". Therefore this command should not be given when there are any files open to <drive1>. Before making the re-assignment all sector buffers are flushed and invalidated to ensure that data does not get onto the wrong disk.

The physical <drive2> must be a drive which is currently supported in the system. However the logical <drive1> can be any of the logical drives "A:" to "Z:", whether or not it is currently in the system. As many logical drives as desired can be ASSIGNed to one physical drive.

If the parameters are omitted then all assignments will be cancelled and if just one drive identifier is given then the assignment for that drive will be cancelled.

This command is like the ASSIGN command in MS-DOS and is provided to enable a program which assumes a particular drive to use another drive. It should not be confused with the MAPDISK command which is intended to support single drive systems.

7.14 MAPDISK COMMAND

MAPDISK <drive1>=<drive2>

Although this command has a similar form to ASSIGN it has a rather different function. If the command:

MAPDISK B:=A:

is given then all references to drive B: will be translated to drive A:, but EXDOS will still consider them as seperate physical drives and therefore as seperate disks. Whenever drive A: or B: is accessed, if it is not the same one as was last accessed then EXDOS will prompt for the correct disk to be inserted (see section 8.2), and will then carry on, checking that it is the correct disk. A MAPDISK can be cancelled by just specifying <drive1>.

Note that <drive1> can only be one of the four internal drive identifiers (A:, B:, C:, D:), but <drive2> can be any drive which is linked in. This facility is mainly provided to support single disk drive systems. In this case the MAPDISK command will be included in the EXDOS.INI file and then commands like "COPY" can use drive identifiers "A:" and "B:" with all the prompting for disk changes being done by EXDOS. The "volume id" allows EXDOS to ensure that the correct disk is always inserted (see section 4.4)

If MAPDISK is used in conjunction with ASSIGN (which is not reccomended as it can get very confusing), then in translating a logical drive number, the ASSIGN is done first, followed by the MAPDISK.

7.15 BUFFERS COMMAND

BUFFERS n

Specifies that "n" "additional permanent sector buffers" (0<=n<=30) are to be used. EXDOS will maintain channel 254 open for these buffers whenever it can. Buffers created in this way will still be allocated after a switch to a new applications program (see section 5.2). Any existing additional permanent buffers will be closed before opening the new ones. "BUFFERS 0" will remove all additional permanent buffers .

7.16 RAMUNIT COMMAND

RAMUNIT <drive> <n>

Specifies that the RAM disk is to be set up using the specified physical drive identifier and with "n" 16k segments of RAM for its data storage. If there are not enough free segments then a .NORAM error will be returned, otherwise the RAM disk will be set up. If a RAM disk was already in existence then the drive number cannot be changed, and it cannot be made smaller although it can be made bigger. If "n" is zero then the currently defined RAM disk will be deleted and the RAM segments freed. It is conventional to use drive "J:" (the last possible drive) for the RAM disk to prevent confusion with extension unit handlers which may be linked in.

7.17 FORMAT COMMAND

The built in FORMAT command will be fairly simple, just to ensure that the system can be used without any disk resident software. It is expected that IS-DOS will be supplied with a more sophisticated disk resident FORMAT program.

7.18 EXDOS COMMAND

This is not a command to be typed by the user, it is provided for programs. To ensure that it is not typed by accident, it is immediately followed in the command string by a <0FFh> byte. It is used by RAM resident extension unit handlers to link themselves into EXDOS, and can also be used to find out the address of EXDOS working RAM to enable IS-DOS to call FISH.

7.19 CD, MD and RD COMMANDS

CD	<pathname>	Change directory
MD	<pathname>	Make directory
RD	<pathname>	Remove directory

The "CD" command specifies a drive and pathname. The whole directory chain must exist, and if it does then it will be made the current directory for the specified or the default drive. The command "CD x:\" will reset the current directory to the root, and the command "CD x:" will display the current directory path for this drive.

The "MD" command specifies a drive and pathname. All directory in the path must exist except the last one which must not, and will be created. The new directory will initially be empty apart from the "." and ".." entries.

The "RD" command specifies a drive and pathname. The directory specified by the path must be empty and if it is will be deleted. If it is not empty then a .DIRNE error is given. The current directory cannot be removed.

8. DISK ERROR HANDLING

There are many errors which can result from using EXDOS. Most of these, such as "file not found" and "disk full" are simply returned to the caller (via FISH, the EXDOS CLI orthe DISK DEVICE) who will normally print an error message.

However there are a set of errors, the "disk errors" which occur in FISH and can frequently be put right by the user. Examples of these are "not ready" (no disk in drive) and "wrong disk in drive". When one of these errors occurs EXDOS prints an error message and then prompts the user for a response which is normally abort, retry or ignore (like MS-DOS). Through FISH, the user can define a routine which can interpret some of these error codes itself (see section 8.1). This facility is normally only available when in IS-DOS.

The exact responses and their meanings vary for different errors but generally "retry" means that the failed operation will be retried, and possibly will result in the error occurring again. "Ignore" means that processing will continue, probably with wrong data and so this response is not recommended.

"Abort" means that processing will return to the higher level with an .ABORT error code, and the real error code will be available in the EXOS variable "DISK_ERR". If FISH was called from the DISK DEVICE or EXDOS CLI then these will stop the command and return the original error code (not .ABORT). If it was called from IS-DOS then any transient program which is running will be aborted and control will return to the IS-DOS CLI (see PER-16).

8.1 USER ERROR VECTOR

The user can set up a routine to intercept these errors before EXDOS prints its messages and prompt, by making a suitable FISH call (see section 3.28). The user's routine will be entered in Z-80 page-3, with the system segment containing the stack in page-2, page-1 un-defined and page-0 containing the page zero segment. This paging must be preserved except for page-1.

The parameters passed to the routine and the results it returns are given below. All other main registers (C, DE, HL, IX) may be corrupted but the alternate register set (AF', BC', DE', HL') must be preserved.

```
Parameters:     IY -> EXDOS RAM area
                 B =  EXOS error code
                 A =  0
Results:        IY -> Unchanged
                 A =  0 => use EXDOS error handling
                      1 => Abort
                      2 => Retry
                      3 => Ignore
                 B =  Unchanged if A=0
```

The routine can make EXOS calls but should obviously not call FISH either directly or indirectly to avoid recursion. If the routine returns A=1, 2 or 3 then EXDOS's error response will be supressed.

8.2 DISK ERRORS and RESPONSES

Any errors which come from a unit handler are put through this mechanism, apart from ones which are interpreted by FISH. These all have straightforward interpretations of the abort/retry/ignore response. They are (for the internal unit handlers):

.ICMD - Invalid unit handler command
.IUNUM - Invalid unit number
.ISECT - Invalid sector number
.NRDY - Not ready
.VERFY - Verify failed
.DATA - CRC error or lost data
.RNF - Record not found
.WPROT - Write protected disk
.NDOS - Not a vaild EXDOS disk

The other errors are generated inside EXDOS. These are:

.WDISK - Wrong disk in drive. This is generated when a disk change operation has occurred while EXDOS has dirty buffers for that disk. RETRY allows the user to put the correct disk back. IGNORE allows it to carry on with the new disk and the data in the dirty buffers will be lost. ABORT stops the operation.

.WFILE - File access of wrong disk. This occurs if the volume id. in an FCB does not match the current disk in the drive. RETRY allows the user to put the correct disk back. IGNORE will carry on with the disk operation on the new disk which is not recommended since it could corrupt the disk. ABORT stops the operation.

.WDRV - Inset disk for drive X:. This error can only occur if the MAPDISK command has been used (see section 7.14). It indicates that the user should put the correct disk for the appropriate logical drive into the actual drive and then RETRY. The operation cannot be ABORTed or IGNORed and so the default handler prompts for an ENTER rather than the usual abort/retry/ignore. Once the user has pressed ENTER, the new volume id. will be checked and a .WFILE or .WDISK error will result if it is not the correct disk.