MEDIA MANAGER SPECIAL EDITION

by

Charles Dillon

Published by DIGITAL PRECISION LTD, 222 The Avenue, London E4 9SE

CONTENTS

1. INTRODUCTION

Thank you for purchasing Media Manager Special Edition. Even Digital Precision Ltd, dedicated as it is to software quality, have rarely spent more time and effort on a program than on this one...

Media Manager Special Edition (henceforward referred to as MMSE) is a very powerful utility to enable QL microcartridges and floppy disks to be "managed" and to recover data (jeopardised by some mishap or mistake) therefrom. Its features include the ability to:

* obtain intelligent directory listings,
* perform selective file operations,
* restore deleted or corrupt files,
* sort directories by name, date, or size, in ascending or descending order,
* read from and write to alien disks (in MS-DOS/TOS formats),
* perform direct sector reading, editing, and writing operations on microdrives, or any disk (alien or QL format)
* obtain full diagnostic printouts of media storage maps for enhanced security,

and much, much more!!

MMSE was produced to supersede the original Super Media Manager, whose brief was somewhat similar but whose implementation was less functional, slower, bigger, rather painful to work with and much less user-friendly.

MMSE will operate on expanded QLs with at least 384K total RAM. It is possible to operate the program on a QL not fitted with a disk interface and drive, if only microdrives are to be accessed. If a disk drive is to be accessed, it must be via a disk interface that allows "direct sector access". A disk interface that is unsuitable because it does not allow direct sector access is the MCS (Micro Control Systems) disk interface. We are not aware of any other unsuitable interface.

We have kept the MMSE manual as brief and to-the-point as possible: we realise that when something has gone wrong and while MMSE is being used in anger, the user is unlikely to want to have to plough through through reams of documentation in order to find out what to do. The design of the program is characterised by absolute consistency of operation: this makes documenting it simple. Once the logic of what MMSE seeks to accomplish has been explained to the user, the steps MMSE takes en route themselves become predictable.

More often than not MMSE is self-documenting as it runs. Sensible prompts are always given, and the type of response required is usually obvious. As such this manual concentrates mainly on the not so obvious points, and any extra information that may be of use to you. In particular there are two sections devoted to the structure and operation of recording methods used for microdrives and disks. These sections will be invaluable to you if one of the automatic or semi-automatic file recovery functions is unable to retrieve your precious data fully. The text will also be of use to those who simply want to experiment, or who want to implement sophisticated protection on their media.

MMSE was mainly written by Chas Dillon and is copyright 1989 Digital Precision Ltd. The machine code work was by Chas Dillon and Freddy Vachha, with some inspiration having been drawn from previous work by Colin Opie. Mission control on MMSE was jointly by Chas and Freddy. Digital Precision Ltd thanks Tony Tebby for some technical backup, and acknowledges the help provided by the reference books on QDOS by Andrew Pennell and Adrian Dickens. All the blame for the MMSE documentation is attributable to Freddy.

This manual is meant to be read in entirety and in sequence. Do not omit anything. Disk-only users should not omit the section on cartridge storage, as many concepts are developed in that section that are vital to aid in understanding the more complex disk storage. If you don't understand how things are stored, a lot of the power of MMSE - especially in the semi-automatic and manual areas of the Sector Editor - will be unavailable to you. There are too many interrelated factors to allow us to completely shield you from complexity. MMSE provides context-sensitive help all over the place, and provides you with a variety of tools. There is usually more than one way of reaching the desired end result.

A key to understanding and mastering MMSE is practice. This manual reports by exception - we decided not to waste paper repeating what is already on the screen. A test program has been provided for you to create example media on which to experiment.

2. WHAT YOU HAVE BEEN SUPPLIED WITH

The files supplied with MMSE are as follows:

BOOT	This fires up the rest of the system.
XTRAS	The run-time version of TURBO Toolkit, which provides various extensions to SuperBASIC required by MMSE. At the time of writing the manual, the version number of the toolkit is v3.20. Do NOT use earlier versions of the toolkit than v3.10, as MMSE requires keywords earlier toolkits don't provide. BOOT automatically invokes the toolkit.
MMSE	The principal MMSE program. Note that this is an executable task (invoked by EXEC and capable of multitasking without any assistance). BOOT, having invoked the XTRAS toolkit, goes on to EXEC MMSE. MMSE is a program written in SuperBASIC and compiled with TURBO. THESE THREE FILES ARE ALL YOU NEED TO RUN MEDIA MANAGER SPECIAL EDITION.
MMSE_DEFAULTS	A data-file containing default values of various parameters MMSE requires (for example, name of the print device). MMSE will look for this file on flp1_ (no matter if you do not have a flp1_), mdv1_ and mdv2_ in turn. If it does not find it, no big deal - MMSE simply uses its own internal set of defaults.
MMSE_SET_DEFAULTS_bas	A SuperBASIC program, started with LRUN, that invokes MMSE_SET_DEFAULTS.
MMSE_SET_DEFAULTS	The configurator program; an executable task which sets up the MMSE_DEFAULTS file.

MMSE_XOVER	An executable task, accessed either directly using EXEC or via the MS-DOS/TOS File Transfer option in MMSE, that permits bi-directional transfer of data between PC MS-DOS disks, Atari TOS disks and QL disks, MS-DOS/TOS FORMAT/DIR and text file translation.
MAKE_TEST_MEDIUM_bas	This program, invoked with LRUN, will create a test disk / cartridge containing 26 / 10 shortish files, to be used for test purposes and for increasing your familiarity with MMSE. The files are "special" in that by looking at any part of the file you can tell which file that part belongs to as well as what the position of the part is in the file.
MMSE_asm	For users who are familiar with M68000 assembler, this is the source code of the microdrive access routines (incorporated in the body of the XTRAS file). Ignore this file unless you are a machine code freak with time on your hands.
MMSE_exts	MMSE_asm + Hisoft Assembler = MMSE_exts. As this file is about 800 bytes long, invoke it with A=RESPR(1024):LBYTES "FLP1_MMSE_exts", A:CALL A if you want to experiment.
MMSE_LINES	This file will be present on the first few disk releases of Media Manager Special Edition. It is a functionally identical but less compact version of MMSE, as - unlike MMSE - it is referenced by line number. In the VERY unlikely event that you are able to get MMSE (the one loaded by BOOT) to crash with an error message, the error message will not contain a line number (it will refer to line 0). This makes it very difficult for us to diagnose the error, which is why we supply MMSE_LINES. If you get a crash and error message with MMSE, reset the QL. Reboot MMSE, and immediately opt to return to SuperBASIC. Then enter EXEC FLP1_MMSE_LINES, CTRL/C into the program and repeat the steps that brought about the crash. You will now get an error message which mentions the line number. Please write to Digital Precision Ltd, describing the precise circumstances under which you obtained the crash, enclosing the medium that "caused" the crash (if applicable - but make a sector-to-sector copy of it first) and QUOTE THE EXACT ERROR MESSAGE. Help us to help you!
BACKUP_bas	The backup program, to be invoked with LRUN. This makes a clone of the Media Manager medium - do this before doing anything with the program!
UPDATES_doc	In the event that we have something to add, we will do so in this Quill file.

This is almost the whole story. There are several wrinkles, however:

(a) MMSE requires there to be a file with a name of length zero to be on the medium from which it is to load ancillary program modules like MMSE_XOVER. The contents of such a file are irrelevant. There is a "zero-filename-length" file on the supplied MMSE medium, and when you use BACKUP_bas to make one or more working copies of the system it will automatically create such a file on each destination medium. For more details see section 3.

(b) If you have received the system on two cartridges instead of disk, the files are spread over the cartridges. Each cartridge will contain its own BACKUP_bas and "zero-filename-length" file. Cartridge 1 will, in addition, contain BOOT, XTRAS, MMSE and MMSE_DEFAULTS. The rest of the files are on cartridge 2.

We MAY add further modules (which will appear as extra files and will be documented in UPDATES_doc) for MMSE in the future - follow our advertisements in the computer press for details.

3. WHAT TO DO FIRST OF ALL

First of all you MUST make a backup copy of the disk or cartridges.

Taking a gamble on your honesty, we have not used any copy-protection mechanism whatsoever within the system.

The best way to make a backup is by using our supplied BACKUP_bas program. Reset the QL and press F1 or F2. Place the master medium in drive 1 and the target for backing up in the other drive. Then enter

```
LRUN "FLP1_BACKUP_bas"
```

and answer the prompts. When the copy is complete, archive the master and use only the backup copy. Make several backup copies. Only use the master for the purpose of producing backups.

If you have the system on cartridge, repeat the process for each cartridge (having substituted MDV for FLP, of course!).

That's it, then. If for some odd reason you do not want to use BACKUP_bas, there are alternatives:

If you have a copy of SuperToolkit (many disk interface ROMs contain one, initialised by tk2_ext or flp_ext:new) you may instead use WCOPY to make a backup copy. There is only one complication: WCOPY will fail to copy our file with name of zero length, so at the end of the copy you should create a "zero-length-filename" file as follows. Enter

```
NEW
```

and then enter

```
SAVE "FLP2_"
```

where the backup copy has been created on flp2_. This applies to cartridge backups too. If you do not have SuperToolkit, and do not want to use BACKUP_bas, you may use the standard COPY command on each file in turn:

```
COPY "FLP1_BOOT","FLP2_BOOT"
```

and so on for the rest (including the zero-length-filename one).

If you only have a single floppy drive, copy the files to ramdisk and back to the target disk (BACKUP_bas will let you do this - it will need to be invoked twice, once for floppy to ramdisk and once for ramdisk back to new floppy). If you have neither a second floppy drive nor a ramdisk, use a microcartridge as an intermediate storage medium (do the files in several lots, with big files like MMSE and MMSE_LINES on their own).

Remember, if you manage to corrupt the system without having made a backup copy, you will NOT be able to use MMSE to recover itself as you will not have a valid copy of MMSE to run!

4. QL MICROCARTRIDGES

The Microdrive cartridge storage format is extremely uniform and compact, employing an endless loop of spliced BASF Video tape travelling at constant speed through the drive. Although the optimum can never be achieved, due to tape splicing and so on, a tape format operation will create a numbered sequence of sectors (irrespective of whether good or bad) up to a theoretical maximum of 255 (0...254). We use the words tape and cartridge and microcartridge interchangeably within MMSE. We use the word 'logical' to denote 'as you would want it' - a logical sequence of sectors would be the sequence in which they contained file data. Physical sequence, in contrast, is the sequence in which they physically occur on the device.

SECTOR 0	Sector header (14 bytes)	+ Block header (2 bytes)	+ Data area (512 bytes)
SECTOR 1	Sector header (14 bytes)	+ Block header (2 bytes)	+ Data area (512 bytes)
SECTOR 2	Sector header (14 bytes)	+ Block header (2 bytes)	+ Data area (512 bytes)
.	.		.
.	.		.
.	.		.
.	.		.

Each sector can be thought of as containing three clumps of information: the sector header, the block header and the data area.

4.1 THE CARTRIDGE SECTOR HEADER

The sector header for each sector is recorded once (at FORMAT time) and once only. It is fourteen bytes long and contains the following data:

Byte offset	Length	Contents
0	1	Identifier byte - always 255
1	1	Sector number (0..255)
2	10	Ten-byte volume label (the name for the medium you specified at FORMAT time)
12	2	Random format number

The volume name is space-padded to the right if the name you supplied is shorter than ten characters long. The random format number in the sector header is used by QDOS in order to determine if the medium in a drive has been changed. As the data exists at the head of every existent sector, the check can be very quick.

4.2 THE CARTRIDGE BLOCK HEADER

The block header for a sector is written each time a data write operation is performed on that sector. It is two bytes long and only has a meaning when the sector in question "belongs" to a file. The block header contains the file number (0..255) and the relative position of the sector within the file (0..255).

The QDOS routines that enable us to read and write sector data also manipulate the block header data. Data and block headers are inseparable in this respect.

4.3 THE CARTRIDGE DATA AREA

Each sector created on a tape cartridge is capable of storing 512 bytes of data. When reading a sector of data, the appropriate QDOS routine will also return the file and block number associated with that sector. Conversely, when writing a sector of data, the appropriate QDOS routine needs to know what file and block numbers are to be associated with that data.

There are three main ways in which the data in a sector is organised, depending on whether the sector in question is the mapping sector, a sector containing directory data, or just a simple data sector. Also, files are stored with their own header data that is related to the corresponding directory entry. These structures are vital to our 'inside' handling of the tapes, so let us look at these next.

4.3.1 CARTRIDGE FILE STORAGE AND BLOCK NUMBERING

Any file that is to be stored on a tape must be stored in physical sectors. So the system splits the file into 512-byte blocks and writes each block, or part block in the case of the last one, into a free sector. Each file saved is given a file number that is used as an index into the directory. Initially at least, saved files are allocated numbers from unity upwards (ie; 1,2,3,...). If for example four sectors are required to save file number six, there will at the end of the operation be four sectors with block headers of <6,0> <6,1>, <6,2> and <6,3> respectively. The directory of a cartridge is handled by QDOS in the same way as a user file (ie; an ordinary file, as distinct from the directory file, map etc). The only difference is that, normally, only QDOS has access to this directory file. All files are stored with a 64-byte header (at the beginning of the logically first sector of the file, making the amount of file data storable in that first sector = 512 - 64 = 448) that contains essential data about the file:

Byte offset	Length	Contents
0	4	Long integer holding file length in bytes This file length includes the header
4	1	File-access byte
5	1	File-type byte
6	8	File information
14	2	File-name length
16	36	File name (Note: Max: 36 bytes)
52	12	"Reserved"

The file-access byte is normally set to zero. The file-type code is 1 for executable programs and 0 for everything else. In this latter case the first four bytes of the file information field contain the default size of the dataspace for that program (stored as a long integer).

Note that the existence of this header means that only 448 bytes of the actual file can be stored in the first sector block (block 0). Any subsequent blocks can contain a full 512 bytes.

Many toolkits use the reserved area for storing a date-stamp on the file (date last written to): this shows up when you perform a WSTAT on the cartridge.

At the end of a cartridge format all non-bad, non-map, non-directory sectors are overwritten with $AA55 byte pairs. When a file is created, and the end of the file does not correspond exactly to the end of a sector (why should it - the odds are better than 200:1 against, even allowing that even-number file-lengths are preferred) the remainder of the sector is filled with nulls (ie; chr$(0)s).

4.3.2 THE CARTRIDGE MAPPING SECTOR

Sector zero is special and is called the mapping sector. It is this that maps logical file storage onto physical sectors.

If you think about it for a minute - the map is what contains the information about where everything else is. Correspondingly, it MUST be in a fixed location or you are in a Catch 22 situation!

Sector zero is split up into 256 two-byte entries. The first two bytes contain the file (F) and block (B) number of sector zero (see table below). The next two bytes contain the file and block number of sector 1, and so on. Because there can never be 256 good sectors (in fact anything over 230 is VERY unlikely), the last two bytes of sector zero are used by the system for other purposes. In practice we need not concern ourselves with these last two bytes.

Map Byte	0	1	2	3	4	5	6	-	-	508	509	510 511
	F	B	F	B	F	B	F	-	-	F	B	spare
Relates to Sector:	0		1		2					254		

In short, the nth and n+1th byte in the mapping sector (n being even, the first byte in the sector being counted as 0) correspond to the n/2th file.

We already know that microcartridge "user" (ie; non-special) files are saved with numbers ranging from unity upwards. There are some other 'files' that the system knows about and which will be found on viewing the mapping sector. We now list the special file numbers in hexadecimal (hex). Hex is denoted by the $ sign; $A=10, $B=11 ... $F=15, and the left hand digit is worth 16 times the right hand one, so for example $FE = 15*16 + 14 = 254.

File number:
- $00 - The directory file
- $F8 - The mapping file (i.e. sector 0, 1 block long)
- $FC - The block is pending a delete operation
  You should never see this at Media Manager Special Edition level.
- $FD - The block (sector) is unused
- $FE - The block (sector) is bad
- $FF - The block (sector) does not exist

If a sector 'belongs' to any of these special 'files' OTHER THAN file number $00 (directory file), its relative position in the file (recorded both in the mapping sector and in the block header) will be shown as $00. This is because the map is always one sector long (and hence its sector has relative position 0) and because the other file numbers ($FC-$FF) are not really files at all. Of course, for file number $FF there will only be a mapping sector entry and no block header - the sector itself (including sector header and block header) just doesn't exist.

Let us examine the two messages the QL produces on formatting a cartridge and then immediately doing a directory operation on it. When a cartridge is formatted, a message of the form 202/206 sectors appears. This means that 206 sectors could be created and 202 of them were verified as being good ones (ie; there were 4 sectors corresponding to file number $FE. If a directory is requested, the medium's title is given, followed by (in this case) 200/202 sectors. This means that out of the 202 good sectors that exist, 200 are left for general file storage use (i.e. 100K of storage). The reason why we are 2 down before even starting is that the directory file (file 0) always exists and starts off just one block/sector long, and of course the mapping sector (file $F8) always exists and is always just one block long.

Special 'files' with file number $FF will always be found, as the mapping sector provides referencing for 255 sectors (from 0 to 254 inclusive) but the physical length of tape packed by Ablex into a cartridge will never be long enough for that many (though heating the tape, or continually running it, will stretch it...). The lowest numbered sector NOT marked as 'belonging to a file of number $FF' is the physically last sector on the cartridge. In the last example, sector 206 will be the lowest numbered one belonging to 'file' number $FF (ie; non-existent). This means that the physical length of the tape is 206 sectors (from sector number 0 to 205, the last non-non-existent one!).

As the cartridge tape whizzes past the read/record head, sectors are encountered in descending order, and very fast too. The QL is set up so that if multi-sector files are being written to cartridge, approximately every thirteenth sector will be written to. If the thirteenth sector is a non-$FD one, it is skipped and a further thirteen is counted down. So if the file was five sectors long (ie; between 4*512-64+1 bytes and 5*512-64 bytes long - 64 is deducted because of the one-off file header) and the first 448 bytes were recorded on sector number 26 on our example tape, the QL would try to write the following 512 bytes onto sector number 13, the next 512 onto sector number 0, the next onto 193 (206-13, there is wraparound - remember, the tape is one endless loop until it snaps). Hold on - we can't write to sector 0, as it contains the map. So the sector sequence might be 26, 13, 193, 180, 167. If sector 180 was bad or occupied, the sequence might be 26, 13, 193, 167, 154. This paragraph does constitute an oversimplication of the process of choosing the next sector to write to - it is really much more complex, to ensure that every possible sector is eventually reached - but this rule of thumb applies a lot of the time.

A curiosity. For reasons not comprehensible to us, the block header for the mapping sector contains $80 (=128) as the file number, instead of $F8. The file number for the mapping sector as given in the map itself is $F8.

4.3.3 THE CARTRIDGE DIRECTORY FILE

QDOS creates and maintains a directory file on each cartridge. It has a file header, just the same as any other file. Every time a new file is saved by a user, an entry is made in the directory file. If the directory file runs out of space in its current block, another sector will be allocated if possible, or a 'directory full' message will appear. Each new entry in the directory file is in fact the 64-byte file header that is also stored at the beginning of each file.

```
FILE 0        (Directory File Entry)          64 bytes
FILE 1              user file                 64 bytes
FILE 2              user file                 64 bytes
  .                    .                        .
  .                    .                        .
  .                    .                        .
  .                    .                        .
  .                    .                        .
```

The important point is that file numbers (below $F8) are an index into the directory file. The position of the directory entry for any user file is given as:

```
file_number * 64
```

Perhaps now you can see why the 'internal' directory file is file zero. The position in the directory file of the entry for the directory file is '0 * 64 = 0'. That is, at the very beginning. We know that at the beginning of every file there is a file header that has the same form as the directory entry. In the case of the directory file, the file header and the directory entry simply happen to be one and the same 64 bytes of data (everything set to chr$(0) except the file length, which goes up in whole sector - 512 byte - increments). If that doesn't shiver yer timbers, prepare yourself for worse!

If you have x "short" (length<=448 bytes) files (including deleted but not overwritten ones) on a cartridge, the number of sectors taken up by the files themselves will be x. In addition, 1+INT(x/8) sectors will be occupied by the directory, and 1 sector by the map. This makes a total of 2+x+INT(x/8). If x=128, the number of sectors required will be 2+128+INT(128/8) = 146. There is no problem fitting this onto the average cartridge, which has over 200 good sectors (typically 210-215). Why is this bad news?

Because the sector (or sectors) belonging to file number 128 will have the same file number in the block header as sector zero (the one that contains the map) - remember the last paragraph of section 4.3.2! Ouch! The map will (correctly) show no overlap as the map's number in the map is 248 (=$F8). Well, how often do you have over 127 files on a cartridge? And even if you do, it is the value in the map that actually determines what belongs to what.

The maximum number of files (obviously short ones) that can ever have been stored at one time on a cartridge which has 254 good sectors is 224 (2+224+INT(224/8)=254). As 224=$E0 is a lot smaller than any of the other special file numbers, 128=$80 is the only problem one.

Incidentally, toolkits do not use the reserved area of each file entry in the directory file to store the date-stamp. This is usually the only difference between the data on a file stored in the directory and the data stored in the 64 byte header at the start of the file. This makes WSTAT quite slow on a cartridge - it has to find the start of each file in turn on the cartridge, which could take upto 7.5 seconds per directory entry.

4.4 CREATING AND DELETING A CARTRIDGE FILE

Let us review what happens when a file is created. A file number is allocated to it, which if multiplied by 64 will provide a pointer to the start of its directory entry made in the directory file. A sufficient number of sectors will be demarcated to store the file, and the mapping sector will have been updated to show exactly which sectors these are and what sequence the sectors have within the logical file. This is performed by storing the file number and block number in the two-byte 'hole' associated with each sector (see Section 4.3.2). The data is written to the sectors themselves, simultaneously updating the block headers with the relevant information.

After you delete a microdrive file (but before you write anything else to the cartridge that might overwrite any sector "belonging" to the file), the 64 byte header at the start of the first (logical) sector of the file remains unaltered. The data itself also remains unaltered. The block header remains unaltered. So far, so good.

The drive map does get altered. The mapping sector entries for the deleted file are returned to the pool of free ones by being altered to $FD00 (ie; file $FD = empty, relative block 0). Also, the first 16 bytes (alas, all 64 bytes if Supertoolkit has been activated, providing a somewhat other than desirable sort of compatibility with the situation which arises when a disk file is erased!) of the directory entry for that file are made into chr$(0)s, which conceals the file from the likes of DIR, WSTAT and WCOPY.

Since the block headers (which tell the world the file to which the following sector belongs, and tell the logical=relative position of the sector within that file) are intact, it will always be possible to automatically "thread" together a file (except for file number 128: we must remember not to include sector 0 as belonging to that file, for reasons already dealt with) that has been deleted from cartridge but not yet overwritten. MMSE makes this very easy.

5. QL DISKS

As disk storage is more complex than microdrive storage, it is advisable to at least read through the previous Section before trying to understand what is going on here with disks.

No one can deny that accessing disks is relatively fast compared to microdrive cartridges. After all, disks are fast-moving random-access devices, whereas cartridges are slow-moving serial-access devices. But the overall speed, although in part caused by an inherently faster device, is also partly due to the way in which sectors are allocated to user files.

Disks are not simply a stream of sectors; there are also tracks and possibly sides to be concerned about. All of these factors give rise to a system that is much more complex than the cartridge system. Also, certain design elements like the need for 'protection' against illegal copying, make the recovery of lost files not exactly trivial. Thus it is not really possible to create fully effective utilities for disks to perform automatic file recovery. We can get close, but that is all.

5.1 QL DISK FORMAT

QL disks are circular wafers of high-quality magnetisable material coated onto a base, invariably on both sides (single-sided disks do NOT exist - the labelling of disks as SS or DS is a marketing ploy). They come in a variety of sizes: 3.5 inches and 5.25 inches are most common, but 3 inches (Amstrad) can also be handled, provided your disk drive is of the right size! QL disk interfaces are very flexible!

The disk size quoted is the edge length of the container - the actual diameter of the disk is marginally smaller.

The good news is that the QL disk format is the same, whatever the size of the disk! Physical dimensions are scaled down, but the layout and logic is absolutely invariant.

Some 3.5 inch drives from Mitsubishi have a rotation rate unsuitable for QL use. Otherwise, almost anything goes!

A disk drive may be single-sided (reading from / writing to side 0 only) or double-sided (reading from / writing to both sides 0 and 1). Double-sided drives have two sets of heads, one for the upper surface of the disk and one for the lower. 5.25 inch disks in single-sided drives can be turned over and the other side used by the heads - it then counts as two disks! The same cannot be done for 3.5 inch or 3 inch disks as their rigid casing is constructed so as to permit only one-way insertion.

With certain toolkits, the insertion of an asterisk as the 11th character of a volume name (the name you specify when formatting it) will cause the disk to be formatted single-sided even on double-sided drives. The drive is simply told not to use its heads for side 1, so all the data gets recorded on side 0.

Each side of the disk is split up into a number of concentric tracks. Each track is like a ring, and the rings are adjacent to one another, with only a small gap separating them. All rings are of equal width. The inner tracks are obviously shorter circles than the outer ones. This arrangement is very different from the format of an audio record (LP, 45, 78) where there is just one groove, a spiral going from the outside rim of the record and gradually curling inwards. The tracks on a disk are all separate.

On a disk recorded double-sided, a track on side 0 plus its positionally-equivalent track on side 1 jointly comprise a cylinder. On a disk recorded single-sided (OK - I give in - we'll call these SS disks, though it is not really the disk which is SS but instead the drive) there is no distinction between cylinders and tracks.

QL disks can be recorded with either 80 tracks per side (numbered 0...79) or 40 tracks per side (numbered 0...39). Track 0 is always the one closest to the rim, and higher numbered tracks are closer to the hub/centre. When a drive is set for 40 tracks, the tracks are more spaced out. If an 80 track drive is used to record 40 tracks by means of some switch on the drive, it usually means that every alternate track will be skipped. Purpose-built 40 track drives tend to record on wider tracks. 80 track drives, whether recording in 40 track mode or 80 track mode, are putting the same amount of data on each track and the tracks in 40 and 80 mode are all of equal width (track lengths vary in both cases depending on how near the centre of the disk the track is - but what we are concerned with is the average): this mode of recording is referred to as double-density. Purpose built 40 track drives give single-density.

Drives that record double density are squeezing more data into unit area of disk written to, and need a higher quality of magnetic coating. In practice all disks should be of sufficient quality to cope with double-density recording. Premium disks can be recorded at even higher densities (quad-density), but QL disk interfaces and drives aren't designed to handle these. Density is an irrelevancy as far as MMSE is concerned. The number of tracks is relevant.

Each track contains nine sectors of equal length. Obviously inner tracks (higher numbered ones) are shorter than outer tracks, so sectors on inner tracks will be shorter than sectors on outer tracks. But all the sectors on any one track will be of the same length.

As the disk is travelling at a constant rate of rotation all sectors (irrespective of the track they belong to) pass next to the head for the same amount of time. The head itself is poised above one track at any particular time and jumps rapidly from track to track as per orders from the interface.

Sectors are physically numbered from 0 to 8 at format-time. Each sector can store 512 bytes of data. So a disk sector stores the same amount of data as a cartridge sector: but there are many more sectors on a disk.

The maximum number of sectors on a tape is 255 (in practice, under 230). A DS/80 (Double-sided, 80 track) disk contains 9 * 2 * 80 = 1440 sectors = 720K. A SS/80 disk contains 9 * 1 * 80 = 720 sectors = 360K, as does a DS/40 disk (9 * 2 * 40). A SS/40 disk contains 9 * 1 * 40 = 360 sectors = 180K.

Good disk drives are DS/80 ones: we supply no others as the loss of storage capacity in the case of the alternatives is immense. Good 5.25 inch drives will have a 40/80 track switch. 3.5 inch 40 track has never caught on, so don't expect to find switches on 3.5 inch drives.

No splicing exists on a disk, and this, combined with factors of reliability and homogeneity, means that (normally) all sectors will be good. In fact it would be most rare if a formatted disk did not return with a full complement of sectors (i.e. 1440/1440, 720/720, or 360/360). If it doesn't, discard it.

Many programs supplied by Digital Precision Ltd (and other suppliers who value their time!) will, when directoried, show denominators other than the full complement. This is NOT because there is anything wrong with the disks! Many toolkits offer a command, FLP_TRACK, which will allow a disk to be only partly formatted. For example,

FLP_TRACK 12:FORMAT FLP1_EXAMPLE

will format the first (starting with 0) twelve tracks of each side of the disk, yielding 9 * 2 * 12 = 216 sectors on a DS drive (half if the disk is recorded SS) - a better name for this extension command would have been FLP_CYLINDER. We format our master disks - including the MMSE system one - so as to have just about enough room to store all the files in. This saves us duplication time. It also encourages good practice on your part, as the master will have so few available sectors on it that you are likely to be deterred from using it as a working disk.... Devious, what? FLP_TRACK 0 sets the drive to its maximum (40 or 80).

There are three main ways in which the data in a sector is organised, depending on whether the sector in question is a mapping sector, a sector containing directory data, or just a simple data sector.

5.2 DISK FILE STORAGE AND BLOCK NUMBERING

Any file that is to be stored on a disk clearly must be stored in physical sectors. In this respect there is little difference between the disk system and the tape system. The disk operating system still splits a file into 512-byte blocks and writes each block, or part block in the case of the last one, into a free sector. The main difference, as we shall see more fully later, is that physical sectors on the disk are allocated to files in groups of three. Therefore, when we talk of a disk allocation block for a file, we are talking about a group of 3 sectors, labelled a, b and c in logical sequence. Each file saved is given a number that is used as an index into the directory. Initially at least, saved files are allocated numbers from unity upwards (i.e. 1,2,3,...).

The directory of a disk is handled by QDOS in the same way as a user file: as with tape, it is file number 0. Only QDOS has access to the directory file. Again this is as true for disks as for tapes. The main difference is that with disks the 64-byte file header (the one at the beginning of the first logical sector of the file) contains the length of the filename and the actual filename. Everything else is likely to be filled with nulls (chr$(0)s, as opposed to "0"s = chr$(48) = chr$($30)) - if not, do NOT rely on it being correct (file lengths are often wrong, for starters). This was done to aid copy-protection methods, but it will prove to be a hindrance to us! Instead, only the directory file (which you will recall contains copies of the true file headers in the tape system) contains the real header data. With disks, this header contains:

Byte offset	Length	Contents
0	4	Long integer holding file length in bytes This file length includes the header
4	1	File-access byte
5	1	File-type byte
6	8	File information
14	2	File-name length
16	36	File name (Note: Max: 36 bytes)
52	4	Date-stamp
56	8	"Reserved"

The file-access byte is normally set to zero. The file-type code is 1 for executable programs and 0 for everything else. In this latter case the first four bytes of the file information field contain the default size of the dataspace for that program (the program may modify its own dataspace as it begins execution). Note that every file has a 64-byte header; it's just that with disks the file copy of the header is almost meaningless. It still means that only 448 bytes of the actual file can be stored in the first sector (sector 'a') of block 0. Any subsequent sectors of blocks can contain a full 512 bytes.

5.3 THE DISK MAPPING SECTORS

It was noted in the last Section that tape systems only use one sector for the storage map. We already know that in disk systems 3 sectors are always allocated per block, and it just so happens that the first allocation block (block 0) is given over to the file allocation map. This gives us 1536 bytes to play with. The first 96 bytes of the allocation map are devoted to information about the disk itself. This leaves 1440 bytes - an interesting number!

Remember that on a double-sided 80-track disk we have 1440 sectors and that each allocation block consists of 3 sectors. This means we can have a maximum of 480 blocks (numbers 0 to 479). It will be of no surprise then to find that the rest of the allocation map is split up into groups of 3 bytes, each group of bytes being used to store the corresponding file and block number for each allocation block (see figure 6). 3 bytes make 3*8=24 bits. The high-order 12 bits are used to store the file number, and the low-order 12 bits store the relative block number (ie; the position of the block within the file).

To find the bytes in the mapping sectors pertaining to allocation block number x, multiply x by 3 and add 96. Let us say the answer is y. Now if INT(y/512) = 0, look in the first mapping sector (track 0, side 0, sector 0). If INT(y/512) = 1, look at sector 3 of the same track/side. If = 2, look at sector 6 of the same track/side. To find which byte to look for, calculate y - 512*INT(y/512) = z. Now the zth, z+1th and z+2th bytes of the chosen sector are the three bytes described in the last paragraph.

As with tape, user files are signified by numbers ranging from 1 upwards, and the directory file is file number zero. By way of example, if file 18 needs 5 sectors to store it, 2 blocks will have been allocated to the file, and the mapping sectors will contain the entries $012000 ($012 & $000) and $012001. Similarly, if there are 25 user files on the disk (assuming no deletions) there will be 2 directory blocks (the first, relative block 0, containing data about 7+8+8 files, with the last two files being detailed in the first sector of the second block) represented by the entries $000000 and $000001.

If the file to which the block belongs is a special file other than the directory, the rules are slightly different.

The map itself is denoted by the single entry $F80000 in the mapping sector, and not $0F8000 as might appear more consistent. Unused blocks are represented by $FDFFFF, and not $0FD000. Bad blocks - rare on disks - are represented by $FEFFFF and not $0FE000. Non-existent blocks (obtained when a disk is formatted single-sided, 40 tracks or by using FLP_TRACK to exclude later tracks) are represented by $FFFFFF, and not $0FF000.

There is a very good reason for doing things this way. While on tape the mathematically-proven maximum number of files is 224, on disk it is much higher. The number of sectors occupied by x user files of short length (in this case, anything shorter than 3*512-64+1 = 1473 bytes will be accommodated on one allocation block) is given by 3 (for the map) + 3*x (for the files themselves) + 3*(1+INT(x/24)) (for the directory). This simplifies to 3*(2+x+INT(x/24)). A value of x=459 yields 1440, so the maximum number of short files that can be saved on a QL DS/80 disk is 459=$1CB. The danger of using the $F8/$FD/$FE/$FF file number convention as for tapes is, of course, that legitimate user files can be expected to have those numbers. Hence the convention is modified for disks. Don't ask us why the map wasn't denoted by $F8FFFF instead of $0F8000 - it is just a convention, and consistency was not a priority. The special file numbers for disks hence are:

File number:
- $000 - The directory file
- $F80 - The mapping file (i.e. sector 0, 1 block long)
- $FCF - The block is pending a delete operation
  You should never see this at Media Manager Special Edition level.
- $FDF - The block (sector) is unused
- $FEF - The block (sector) is bad
- $FFF - The block (sector) does not exist

We value consistency and use only the first two bytes of the file number of special files in MMSE narrative - this keeps the conventions the same as for tape.

At the end of a disk format all non-bad, non-map sectors are overwritten with $30 bytes ($30=48 corresponds to the character zero: this is quite distinct with filling with nulls = $0 bytes, which is quite another thing). When a file is created, and the end of the file does not correspond exactly to the end of a sector (why should it - the odds are better than 200:1 against, even allowing that even-number file-lengths are preferred) the remainder of the sector is filled with nulls. If this sector was not the c sector in the allocation block - ie; if there were other sectors to follow in that allocation unit - then their contents will not be disturbed. So if a new file is created on a kosher disk from which no deletions have been made, and the file contains 10 capital "A"s, the actual allocation block would contain a 64 byte header (containing nothing very useful) & 10*chr$(65) & 438*chr$(0) & 2*512*chr$(48).

Note that this allocation mechanism explains the two messages the QL produces on formatting a disk and then immediately doing a directory operation on it. When a disk is formatted, a message of the form '1440/1440' sectors appears. This means that 1440 sectors could be created and all of them were verified as being good ones. If a directory is requested, the medium's title is given, followed by (in this case) 1434/1440 sectors. This means that out of the 1440 good sectors that exist, 1434 are left for file storage use (i.e. 717K of storage). The reason why we are 6 down before even starting is that the directory file (file 0) always exists and starts off just one allocation block (3 sectors) long, and of course the mapping sector (file $F80) always exists and is always just one allocation block (3 sectors) long.

The 96 bytes of disk information stored at the beginning of the mapping block are used in the following way:

Byte offset		
$00	4 bytes	format ID 'QL5A')
$04	10 bytes	medium name (right-space-padded)
$0E	2 bytes	format random number
$10	4 bytes	count of updates
$14	2 bytes	free sectors
$16	2 bytes	good sectors
$18	2 bytes	total number of sectors
$1A	2 bytes	sectors per track (normally 9)
$1C	2 bytes	sectors per cylinder (9 or 18)
$1E	2 bytes	number of tracks (40 or 80)
$20	2 bytes	sectors per block (normally 3)
$22	2 bytes	block number of dir EOF
$24	2 bytes	byte number of dir EOF (0 to 511)
$26	2 bytes	sector offset/track
$28	18 bytes	logical-to-physical sector table
$3A	18 bytes	physical-to-logical sector table
$4C	20 bytes	(spare)

Note that near the end of this 96-byte block there are two 18-byte tables. It has been hinted at already that certain things go on in the background of a disk filing system to keep things fast. These two tables are precisely for this purpose.

5.3.1 THE DISK TRANSLATION TABLES

To keep sector accesses to disks as fast as possible, adjacent logical sectors are stored, spaced apart, in actual physical sectors. The two tables in the 96-byte descriptor define the logical-to-physical relationship. If we take a double-sided disk as an example, there are 18 sectors per cylinder, which will be used in the following order:

	a	b	c			
sectors	0	3	6	side 0	-	allocation block 0
	0	3	6	1	-	1
	1	4	7	0	-	2
	1	4	7	1	-	3
	2	5	8	0	-	4
	2	5	8	1	-	5

This means that our mapping block (block 0) actually resides in sectors 0, 3 and 6 of side 0, track 0. Moreover, the 96-byte disk definition table will be found at the very start of sector 0, track 0, side 0. And this latter fact will remain true no matter what type of disk it is (i.e. 40 or 80 track, single- or double-sided). This is clearly important, so the disk filing system can tell very quickly and reliably what type of disk it is trying to handle.

On a disk system, the first cylinder (ie; track 0, sides 0 & 1 or just side 0 if the disk is formatted SS) will often be devoted to the map and the directory (which, in Track/Side/Sector terms starts on 0/1/0, 0/1/3 and 0/1/6 unless these sectors are damaged, or on 0/0/1, 0/0/4 and 0/0/7 if the disk was formatted SS). The way in which the QL decides where to fill in the next block on a disk where there are "holes" - caused by deleted files - is very complex.

When moving across tracks, a further timing factor is introduced: the sector offset per track value stored at offset $26 in the definition table. Each time a track is traversed, a logical/physical sector table, such as that shown above, will be modified. Each element in the table, in terms of sector numbers (not sides), is modified to be:

```
(original_entry + (track * offset)) MOD sectors_per_track
```

This offset per track is hardwired in as 5, and therefore we can see that allocation block 6 will in fact reside in sectors 5, 8, and 2 (in that order) of track 1, side 0. It is important to keep track (sic) of these physical sector allocation tables in order to make sensible attempts at file retrieval. You will be pleased to hear that Media Manager Special Edition is extremely helpful in terms of letting you know exactly what belongs where!

Compare this with the relationship between physical and logical for tape, where a logical progression of +1 sectors often corresponds to a physical shift of -13 sectors (with wrap-around).

5.4 THE DISK DIRECTORY SECTORS

These are much the same as the cartridge ones (except that sectors are allocated to the directory, as with any other file, in clusters of 3). The directory entry for the directory itself is 64*chr$(48).

5.5 CREATING AND DELETING A DISK FILE

Let us review what happens when a file is created. A file number is allocated to it, which if multiplied by 64 will provide a pointer to the start of its directory entry in the directory file. A sufficient number of allocation blocks (each of three sectors) will have been demarcated to store the file, and the mapping sector will have been updated to show exactly which blocks have been used. This is performed by storing the file number and block number in the three-byte 'hole' associated with each allocation block (see Section 5.3). The data is then written to the allocation blocks themselves; within each allocation block this is always in the sequence a b c.

Unfortunately, QDOS is a bit more enthusiastic when deleting a disk file than it was deleting a tape file. While the file contents themselves are left untouched (until an overwrite occurs), mischief is created in the directory. On tapes only the first 16 bytes of the directory entry are filled with nulls - we are left the name. On disks the entire 64 byte directory entry is filled with null characters when a file is deleted.

What makes this even worse is that on tape you had two copies of the file header, one at the start of the file and the other in the directory. On disk there was but one decent copy of the file header, and that was the copy in the directory. Note the past tense. The one at the start of the file itself is good only for the filename and the length of the filename - and you have got to find it first! Also, on tape you had a block header which gave independent information about file ownership and relative position within the file - there is no such mechanism at all on disk.

There is one small redeeming feature possessed by disks in this respect. When the mapping block entries for the erased file are returned to the pool of free ones by having the most significant byte of the 3-byte record set to $FD. So if for example we delete file number 6, which is, say, 2 blocks long, the two mapping records will be altered thus:

```
$006000 -> $FD6000
$006001 -> $FD6001
```

We have no name in the directory, disk sectors do not have headers that tell us which files they belong to, and the actual file copy of the file header is incomplete. However, the mapping block entry has not been totally obliterated, and the first half of the second byte of the entry for that block preserves some of the information relating to the owner file number. The whole of the relative block number is preserved.

If the disk contained more than 16 files, it is possible that there will be more than one entry such as $FD6000. For example, one could refer to deleted file 6, and another to deleted file 22. Without patient and careful scrutiny of map entries to physical sector contents there will be no way of knowing which is which.

6. CONFIGURING MMSE

This is usually a one-off process. We already supply MMSE with sensible default values, but you may want to change them.

	Supplied Default	What it is
System Device	flp1_	The device from which MMSE ancillary programs will be loaded.
Print Device	ser1	The device to which reports will be output: note this can be set to be a file.
Baud Rate	9600	Baud rate of Print Device. Value irrelevant if the printer is a parallel one or if the Print Device is a file.
Output Device	flp2_	The device on which the collector file (the one used for collating sectors taken from the damaged Working Device) and other user-created files (except reports which go to the Print Device) will reside.
Collecter name	COLLECTOR	The name of the file used for collating sectors from the damaged Working Device.

If we supplied you with the MMSE system on cartridge, we will have altered the supplied defaults to more appropriate mdv values.

If you have a non-FLP disk interface (Microperipherals), contact CARE Electronics for a QFLP ROM upgrade. Alternatively, use the VSET command before booting up the MMSE system.

The Output Device should be a file-type device (ie; mdv/flp/win), not a printer.

There is no harm in setting the Print Device to the same file-type device as the Output Device and/or the System Device. Don't blame us if you clutter up your working copy of MMSE!

To configure MMSE, boot up the QL, press F1 or F2 and then enter

```
LRUN FLP1_MMSE_SET_DEFAULTS_BAS
```

Then follow the prompts. At the end, a new MMSE_DEFAULTS will be produced, containing the new default values for the main MMSE program.

If MMSE was supplied to you on microdrive, put cartridge 2 in drive 2, cartridge 1 in drive 1 and enter

```
LRUN MDV2_MMSE_SET_DEFAULTS_BAS
```

Remember that MDV1 is the device containing MMSE_DEFAULTS.

7. USING MMSE

Boot up your QL with the MMSE backup disk in flp1 (or cartridge 1 in mdv1): MMSE will load automatically.

It is always best to fire up MMSE from a reset, ie; on a clean machine. This is because the operating system creates slave blocks and makes assumptions about things not changing that are invalid where direct sector access games are being played.

7.1 THE MAIN MENU

You wil be presented with a screen bearing the title 'Media Manager Special Edition vX.XX - Main Menu'. Twelve menu option windows will be visible - some of them will be empty.

At this stage you can remove the MMSE medium. You will only need it again if you want to load the MS-DOS/TOS File Copier (or other ancillary add-ons to MMSE that we may release in the future). The device from which you booted up is fully available for whatever use you wish to put it.

The four corner option windows contain text in white. Other option windows contain text in green. The white text indicates that those options are available - ie; selectable. The green ones currently aren't. The blank ones are just there to get the total up to 12!

At the foot of the screen there is a two line message:

Use cursor key to Move, ESC to 'end', SPACE ¦ ENTER to Select
Define working, output and print devices

Look carefully at the option windows. The top left one, 'Select Primary Devices', has a green border superimposed around it. That green border is your selection cursor. That is the current option. The bottom line of the screen ('Define working, output and print devices') tells you what that option offers. This on-line help, present almost all the time, makes the task of this manual much simpler. Just read the screen.

MMSE has a number of similarly designed menu screens, which is why we are describing this in detail.

As a rule, when you enter a menu, MMSE puts the selection cursor on the option window it thinks most appropriate for you to select. The executive action is up to you.

The penultimate line on the screen tells you how to navigate the selection cursor and how to select items. It all sounds simple. Try to move the green border using the arrow keys. It won't move.

This is because MMSE is a multitasking program, and we have not as yet transferred keyboard control over to it. The flashing cursor at the lower left of the screen (SuperBASIC window #0) shows us that control is still with SuperBASIC. Enter PRINT 2+2 to prove this.

The way to toggle control in and out of MMSE is by using CTRL/C - hold the CTRL key down and tap C. You will notice a cunningly concealed cursor at the top left of the Main Menu will start to flash. MMSE is is now listening to your keyboard commands, and moving the green border selection cursor around is easy. Try left, right, up and down repeatedly until you understand the underlying mechanics of the screen. Why bother with Up and Down when Left and Right wrap-around so well? Try pressing number keys to jump to option windows.

Note that we allow you to move the cursor over blank option windows and over the currently unavailable, green-text options. However, if you pressed Space or Enter in order to select them, your command would be ignored.

Note how the help line at the foot of the screen changes - it is context-sensitive, and relates to the option over which you have placed the selection cursor (including currently non-selectable ones).

If you press ESC when faced with a menu screen, you will be sent to the exit option - usually the bottom right hand option window: you will then have to press Space or Enter to execute the option. If you press ESC in reply to a query, you will escape automatically back whence you came. This functionality applies all over MMSE.

On this Main Menu screen, ESC gets you to the Return to SuperBASIC option. This is not too sensible at this juncture! Select it carefully, pressing Space or Enter just once.

MMSE now gives you an 'Are you sure' type get-out question, to protect you against the consequences of an inadvertent keypress. In this case the question is 'Quit Media Manager (y/n):'. / denotes a choice by you. When MMSE presents you with such a choice, the pressing of any non-ESC, executive key (ones like SHIFT don't count) other than the second choice will be equivalent to pressing the first-named key (in this case, y) . So here, pressing J will the same as pressing y. Don't. Press N - there is more to see in MMSE (that rhymes). Of course these questions are NOT case-sensitive - we don't expect you to keep a running track of the condition of your CAPS LOCK key! The ESC key could also have been pressed - in this case, its effect would have been as for N. As stated earlier, ESC will get you back to the state you were in before you committed yourself to the course of action which prompted the question.

The results of your addition exercise from SuperBASIC are probably still on the screen. Pressing F4 from within MMSE will tidy up the screen.

You can CTRL/C in and out of MMSE at will, except when it is the middle of certain operations (like tape formatting, when it can't afford to listen to you as timings are critical). This means that you have all the power of SuperBASIC at your beck and call at virtually any time. Even if the screen gets totally overwritten/cleared, you can CTRL/C back into MMSE and restore order with F4. The position of the flashing cursor tells you where control resides.

Select the main menu option 'Set Machine Date': it is always a good idea to use date-stamping of files, and it is no use date stamping with the randomly generated QL power-on date. Notice how, on completion you are returned to the Main Menu with the selection cursor still on the same option (permitting easy reselection by simply pressing Space or Enter again - this is found in many places in MMSE).

Choosing the option 'MS-DOS/TOS File Copier' will cause the MMSE_XOVER ancillary program to be invoked - its operation is fully described later in the manual. Explore this route. Reply with Enter to the frst few questions/prompts - you will then see the following

```
System device to use: flp1_
```

This is asking you to confirm where MMSE_XOVER is to be found.

If you changed the default system device during the configuration process, that is the name that will appear after the colon instead of flp1_. Note the position of the flashing cursor at this juncture. If you press Enter, the value of flp1_ is accepted. If you use the arrow keys, you can edit the string (holding CTRL down too if you want to delete characters) and, once satisfied, accept it with Enter. Or you can press ESC to escape, which is what we suggest at this exploratory stage.

It is at these points - when input is prompted for - that F4 will not refresh the screen.

Again, this facility of providing you with an editable default, accepted with Enter and escaped from with ESC, is common in MMSE.

The remaining white text option on the Main Menu is 'Select Primary Devices'. Until this is done all the other (currently green text) menu options will remain unselectable. That is why we put the selection cursor on this option when you entered the Main Menu.

Choose 'Select Primary Devices'. You will be presented with a subsidiary menu, called 'Primary Device Selector'. The rules are of course the same. Play around examining system settings. With typical friendliness, MMSE allows you to adjust at run-time the configured defaults (which, after all, only represent suggested values you think most likely to represent your needs - exceptions were expected) for Output Device and Print Device. Such run-time adjustments, unlike ones done with the MMSE_SET_DEFAULTS configurator, are forgotten when the machine is switched off.

The selection cursor is on the menu option window 'Select Working Device', and we would be right in deducing that this is the one we need to choose. Working device is not one of the presettable defaults. The working device is the one on which the "target" medium is to be found - we may be attacking the target because it has become in some way corrupted or damaged, or because we want to adjust or tidy it, or simply because we want to see MMSE at work. If we are engaged in any sort of data repair or recovery work, the Working device should be DIFFERENT from the output device as, by and large, we will want to write our recovered files (or batches of concatenated sectors) onto a new, good medium!

Let us prepare a test medium for MMSE. CTRL/C into SuperBASIC and LRUN the MAKE_TEST_MEDIUM_bas program. Create a test disk (later on, repeat this with a created cartridge) on a blank floppy - let us assume this is in flp1_. CTRL/C back into MMSE and ensure that the Working Device name is flp1_, pressing Enter to confirm. You will be asked whether you are intending to write to the working device. Let us say yes (press y or Enter etc).

What MMSE will now do will entirely depend on whether the working device selected is a disk or a cartridge. Because of the inherent unreliability of cartridges, MMSE makes an image of the working tape in RAM immediately you select tape as the working device. It does this by reading in all the existing sectors (complete with sector headers and block headers) on the tape - the sector number being scanned flashes up on the MMSE screen. Long pauses at certain sector numbers usually means MMSE is having difficulty in reading that sector. The process of creating a RAM image can take from as little as 20 seconds on a healthy cartridge to 10 minutes on a battered one.

If the specified working device was disk, MMSE first scans the map and presents the number of Tracks, Sides and Sectors per track it has gleaned from the map. You are allowed to edit the first two items if you disagree with the value read-in. You would generally only choose to do this if the map was corrupted or if the map was purposely slugged (copy protection being the goal) in order to conceal/understate the true size of the disk.

The next screen will appear for both tapes and disks - the Volume Details screen. A lot of information gleaned from the map and directory will be presented here. More important, a health check will have been performed on both map and directory. No news here means good news. Take careful note if either the directory or the map are reported to be unhealthy - what you should and should not now do is governed a fair deal by these conclusions.

Pressing a key will allow us to go on.

By this stage you may have noticed that MMSE permits type-ahead. This makes MMSE very fast to use once you are familiar with the program. In most cases you are protected against the consequences of indiscreet typing ahead by questions with "safe" default answers. Go slowly at first.

We have covered this first screen in GREAT detail. This is because a whole host of concepts here (white/green availability indicators, cursor movement, ESC, F4, CTRL/C, menu hierarchies, default direct key responses, default string editing, case insensitivity, type-ahead, run-time modification of preset defaults etc etc) are common to the whole of the package. We won't boringly repeat any of this. Make sure you master things so far before you proceed. Re-read 7.1!

Let us now opt to return to the Main Menu, where - surprise surprise - all options are now presented in white (they are all accessible). We will examine them in turn, treating disks as the main case and commenting on tape only where there are differences. But first let us briefly examine what the source of medium problems might be.

7.2 WHAT GOES WRONG WITH DISKS AND CARTRIDGES

Prevention is better than cure. Make a backup. Make many backups if you are using tape. There are so many ways that a file could go astray, and quite a number of ways that MMSE can get it back. With all these permutations staring at us, it is worth stepping back and seeing just how file recovery can be systematically handled.

Clearly microdrives and disks will have to be handled differently during the final stages, but we can still cover file recovery techniques using Media Manager Special Edition without becoming too involved in media detail.

Problems could have arisen in a number of ways. Cartridges go wrong of their own accord. The medium could have been damaged by some physical accident (orange juice is worse than coffee). A rogue program could have overwritten parts of a medium. Faulty hardware could be to blame (we "fondly" remember an early disk interface - v1.06 we think - which trod heavily on the directory/map if a file of zero length (not a zero-filename-length) file was encountered...). A glitch in the mains supply, an inadvertent power-off or a loose drive cable could have wreaked havoc in the middle of a medium access.

If a 5.25 inch disk is damaged/bent, cut open the sleeve carefully and release the disk, which will possibly be readable. You can't do this with other disk sizes.

Damage done to Archive files by illegal exit from that program are best fixed using the dedicated PDQL program 'Recover', which concentrates solely on Archive files. Special Archive info has to be put onto the file to make it a legal one.

A damaged Quill file with sectors missing will not be readable into Quill. After recovery with MMSE, load the file with RU into Editor Special Edition, delete the control info from top and bottom, use F3 followed by T; RP E.x..;S where x represents the character obtained by holding CTRL down and tapping the pound key. Then set a right margin at 80 or whatever and enter F3 and then T; RP PR . Save the file and load it back with R. A bit of minor editing and you are home and dry. If you really want, you can load the cleaned-up ASCII file into Quill with import.

7.2.1 DELETED FILES

This is probably the easiest type of file to recover. The important point is that no save operation should have been performed between the time that the file was deleted and the time when you try to recover the file. If such a save operation has been performed, there is good reason to assume that the original file's directory space and some or all of its allocated blocks have been overwritten by the new file.

Obtain an unfiltered MMSE directory listing of the medium: it will show the deleted file's file number, and your best course of action is to use the Recover Deleted File utility, which peeks at block headers. If the deleted file was from a disk, you won't be allowed to use this utility: disk sectors do not have block headers to say which files they belong to. Advice will, however, appear on-screen.

When a deleted file is recovered, automatically or otherwise, it is in units of whole sectors (or 3-sector blocks in the case of disks). As such there may be some garbage hanging around at the end of the file. MMSE provides you with the opportunity of truncating it.

However, if the file was a SuperBASIC program, you could simply load it. The SuperBASIC loader will not parse the garbage at the end, and you will either have a totally good program or a program with some extra lines at the end containing the keyword MISTake. Simply delete any spurious lines at the end of your program and save it back onto the medium in question.

The four Psion programs are also likely to ignore garbage that exists after the end of their 'known' data files. So the next time you use the text or data file and re-save it from within the appropriate Psion program, the garbage will disappear.

A problem clearly arises, however, if some terminating garbage is not ignored by some other program. At this point you will need to use a sector editor to shorten the length of the file in question, so as to eradicate the rubbish. You do this simply by altering the file-size parameter in the directory entry. With microdrives you should also change the copy in the actual 64-byte file header. The true file size can be obtained from a previously taken Media Manager Special Edition directory listing. If you do not have such a listing, you will have to find out the hard way, by using map data and the sector editor to find out where the end of the file really is. Of course you could always make an estimate of the file size. If you cut off too much, you can always increase the file size again - the file data is not removed when a file is shortened in this way. You can take comfort from the fact that the existence of end garbage rarely gives rise to a major problem.

An excellent program to use for this sort of thing is Special Edition Editor, which is capable of loading and manipulating ANY file including ones that contain unprintable characters.

7.2.2 BAD OR CHANGED MEDIUM

This is the horror-story error message that normally sets a shiver going down most users' backs. It is almost certainly caused by a faulty (bad or improperly changed) sector somewhere in the storage area of the file you are trying to access.

The first thing you must do is make a Sector to Sector copy of the medium on a new, clean, problem-free formatted medium.

The easiest way out of this problem is to use MMSE's utilities for recovering lost/corrupt files and writing them out to another medium. These utilities will recover all the good sectors and allow you to specify a 'fill' character to replace each byte in any bad sectors. In this way you obtain a readable file that could be patched later on.

Any report given on map/directory health (back when you specified the working device) may alert you that the bad/corrupted sector belonged to the map or directory. Alternatively, if you use the sector editor to scan the sectors belonging to map and directory, and if the contents of one or more of them shows up as starting with <TE> (for transmission error) you know that there is trouble there. Checking the filtered and unfiltered directories may visually inform you if the directory is corrupted, though caution is recommended before you arrive at aconclusion - perhaps a corrupted map is causing you to look at sectors which have nothing to do with the real directory.

If the sector causing the problem was a directory sector, use the Rebuild directory utility which constructs a new directory from the map information, optionally saving the old directory as an ordinary user file on the working device.

If the sector causing the problem happens to be a mapping sector, your problems are more acute (particularly so with disks). You can still use MMSE to re-generate your medium (onto another medium, though). It will take time, and you will normally use all the editing, movement and string-search facilities available to you. Try to recover things a file at a time and double-check all your editing activities. Spend as long as it takes with the sector editor - which has all sorts of superb movement commands - to view the medium, in order to ascertain exactly what has gone wrong. Time spent checking in this way is always worth it - it will stop you from doing more work than you have to! If a re-generation job is necessary, make sure that you create the recovered files on a fresh medium and NOT on the bad medium!

In general, the best approach is to obtain a hard copy of the mapping block data (not as raw data, but as a Mapping Table which shows the allocation block and a-b-c value for each sector on the medium, and which shows to which file the sector belongs and what is its position within that file). This will be of use unlss the map has been totally corrupted. Also, get a printout of the Allocation summary, which will show the contents of the first and last 64 bytes of each allocation block (data is ALWAYS recorded in a-b-c sequence within the sectors of the allocation block itself, so individual sector information here would be unnecessary) and repeat map-derived data re alleged ownership of the block. You now know the physical layout of the disk.

Now, using the string Search (finds anything) and Locate (finds filenames - ie; only searches in the 36 byte area starting at offset 16 from the start of an allocation block: note that this means it will also find things that aren't filename related if the allocation block being scanned didn't have a file starting in it) options in the sector editor, you can resolve any "doubtful identity" cases where the first/last 64 bytes of a block did not help identify it. Use the sector editor navigation facilities to confirm this suspicion. Turning back to the hard copy of the mapping sectors and the sector editor itself, make a list of any entries of the form (in hex):

where 'x' is the least significant nibble of the appropriate file number. When you are sure that you know which entries belong to your file, update the mapping block appropriately (i.e. replace $FDx with the true file number, for example $006). You can then return to the Recove lost/corrupt file procedure and let this collect the file for you and place it onto the output device. Finally you should re-format or discard the current disk.

7.2.3 NOT FOUND

Media which give this message when a DIR is attempted from SuperBASIC have either got a corrupted map or corrupted directory or both. Treat these as you would a bad or changed medium situation where the map and/or directory are corrupt.

7.2.4 OVERVIEW

If something is wrong with a medium, first make a Sectorr to Sector Copy of it and only work with the copy. That way, errors made by you while using MMSE do not turn a mishap into a tragedy.

An important point to remember is that you should always start at the highest (ie; most automatic) level for file recovery. Try options in the Utilities menu before you contemplate moving "down" to the next level, the sector editor. Spend as long as it takes to find out what exactly has gone wrong. You do not want to do more work than is necessary.

While you are working on a device (the Work device) you will get an 'in use' message if you try to access it from SuperBASIC. The way to 'free' a working device is to select another working device or to quit the program. Do not attempt to change the medium in the working device without going through the primary device selection.

Where a repair job has been done at sector level by rewriting to the original faulty device (usually not good practice, but OK if the corruption was very localised and the underlying medium was undamaged and writeable-to), do not panic if on return to QDOS there appears to be no change. QDOS may not have noticed (due to its internal buffering and reliance on the random format ID) that the program has changed the directory etc. In order to see (say) your un-deleted file appear once again in its full glory, reset the QL and then perform a directory: this forced QDOS to abandon its preconceptions about the state of the medium!

Having done a repair job on tape, remember that all the work was actually happening in the RAM image area. Some disk work is also done in RAM. When quitting from MMSE, or when selecting a new working device, MMSE will check if you want to discard the changes or save them back to the working device.

7.3 SECTOR COPIER

This Main Menu option makes a clone of the working device, even if that device is bad and even if it contains unreadable sectors.

You are VERY VERY strongly recommended to use this option before you make ANY attempt at recovery. Even if you are not going to do the repair directly on the working device, make this backup.

Don't bother to make a Sector copy if your medium is good and all you want is to carry out some management. Of course you should have made a backup - by conventional means, like WCOPY - of the intended working device before you attempt any manipulation of it under MMSE. All we are saying is that you needn't make a Sector copy... we aren't suspending normal good practice! Do not discard the original until you have checked (on a clean machine - remember QDOS's internal buffering) that the 'managed' backup is OK. OK doesn't just mean that the directory comes up fine - COPY a file or two to SCR, or invoke a program on the medium to convince yourself.

Back to Sector Copying.

You are not asked for target device name. This is because the target must be of the same TYPE as the source (= working device). So if your working device is flp2_ we will use flp1_ as the target, if your working device was mdv1_ we will use mdv2_ etc.

Once done, archive the original and place the clone (which is an exact clone, down to random format ID - please do not use this for piracy!) in the working device. There is no need to reselect the working device as the two media are perfect clones (this is the one exception to the rule about not changing working device behind MMSE's back, as given in section 7.2.4).

If your medium is tape, make sure that the target has at least as many total sectors and as many good sectors as the source. If you can't find such a tape, all is not lost: don't make a sector copy. As we will be working from the RAM image anyway, there will be no wear and tear on the original. In this event (inability to make a tape sector copy) perform the recovery onto the output device and do not patch the original (do whatever you like in RAM, but do not download the patched work onto the original, unbacked-up tape).

7.4 DIRECTORY MANAGEMENT

This is ONLY for use on media that are OK! Use it on a corrupted medium and all hell could break loose....

We leave it to you to examine and experiment with the varied useful options within this sub-menu: the headings are self-explanatory and the context-sensitive help is conclusive.

Directory changes are not "saved" to the working device (or, in the case of tape, to the RAM image) until you choose to Write Directory.

Do not select Volume Management. It will attempt to load a utility program module called MMSE_VOLUME_MANAGER which does not exist. If and when we decide to expand the functionality of MMSE, we will offer MMSE_VOLUME_MANAGER (for nominal cost to MMSE owners) via our press advertising. MMSE has been built with system expansion in mind.

Beware of QDOS internal buffering if you return to SuperBASIC after having performed the directory manipulation: the operating system may not realise that the medium has been internally changed.

7.5 UTILITIES

This sub-menu contains the sort of commands you will want to use if something has been deleted, lost or corrupted, but the map itself is intact. The executive (Recover/Rebuild) commands here provide a degree of automation to the recovery process. The information commands (the 4 Show ones) are very useful, and - unless the medium treatment is fully automatic - it is wise to print out the mapping table and the allocation summary.

Do NOT Recover/Rebuild onto the working device - use the output device (which can easily be amended by ESCaping back to the Main Menu and choosing to Select Primary Devices) instead. Recover makes a guess at file length (remember, all it knows is that the file ends SOMEWHERE in a particular block - its guess is the prudent one, that the file goes right up to the end of the block) which you are allowed to edit. You may choose to use the Sector Editor to determine the exact end-point. If the file to be recovered has missing sectors, you will be prompted for a "fill" character - choose something sensible like '?'.

Information is generally given in decimal rather than hex within this section, except in the mapping table where a hex representation allows easier understanding of file numbers (remember how deleted files are marked in a disk map). Common-sense abbreviations for Side (Sd), Track (Tr/Trk), Sector (Sc) and Block (Blk) are used. Special file numbers are highlighted with an Sp. Generally speaking, all numbering is from base 0 (consistent with all that we've said earlier in the manual) and not base 1. When "describing" a sector, the letters a/b/c after an allocation block number denote the logical sequence in which the sectors will have been written to within that allocation block. Note the distinction between block (or relative block) which is an indicator of relative logical position within a file, and allocation block, whose value shows the physical position within the file and whose value can be used as a reference pointer into the map to determine file ownership (and relative block number).

In the allocation summary, the entire range of characters in the ASCII set (0-255) may be encountered, including unprintables about which your printer would say unprintable things if your printer could speak. For this reason, we represent the null character 0 ($0) by a _ , the character 255 ($FF) by a . , all characters in the range 1 ($1) to 31 ($1F) by a down-arrow and all characters in the range 127 ($7F) to 254 ($FE) by an up-arrow.

If you have assigned a file instead of a printer as the Print device, you can examine and manipulate the mapping table (the result of processing/analysing the 'raw' information in the mapping sector(s)) and allocation summary files using any ASCII Editor. The best one is Special Edition Editor from Digital Precision, whose programmability will allow you to fully process the data if you so wish.

As before, the best way to familiarise yourself with the options here is to try them out. Use the MAKE_TEST_MEDIUM_bas SuperBASIC program to make a test disk and also a test cartridge. Do a DIR on each from SuperBASIC, and COPY a few of the files on it to the screen (using COPY "FLP1_FILE:A", SCR or similar) to figure out what you expect to see in the files. Produce a printed allocation summary, mapping table and unfiltered directory (the filtered directory, which was available in the Directory Management sub-menu, is less useful as it, like the SuperBASIC DIR, removes deleted files and removes 'blank' entries that will pad out the allegedly unused part of the last relative block of the directory file) and reconcile the information in the three.

In doing this, make numerous references to the explanations and tables given in sections 4 and 5 of this manual. If you spend less than two hours on this, you will certainly have missed things.

Also view the Volume characteristics (the same screen that you got once MMSE finished checking the newly-specified working device): especially the information on the directory EOF. Later on, use the Sector Editor and physically examine the "raw" map sector(s) and the directory files (whose location can be found by scanning the allocation summary or the mapping table). Print out their contents and reconcile these - continually referring to their structures as defined in sections 4 and 5 - with the 'processed' values obtained from the allocation summary, mapping table and unfiltered directory. Allow at least two hours for this. In your checking, see the relationship between file length deduced by counting entries for the file in the map-derived data (the raw map itself, the allocation summary and the mapping table) - remembering each entry comprises a sector or allocation block - with the file length according to the directory file. It is a bit complex. Persevere - you'll get the hang of it, and then you will be in a position of power!

A No Name filename in the show unfiltered directory option represents a directory header where the filename-length is illegal (>36). Equally, absurdly long file-lengths are shown as >=1E6.

Incidentally, you should already know where the map is from the earlier sections - its position never changes. If while formatting a disk any of the three map sectors give the result 'bad', the format aborts with a format failed message immediately. To see this happen, try a format without any disk being present: it doesn't take long to happen, once the interface abandons its attempts to retry. On cartridge sectors are marked at format time, so the first non-bad sector that is found is marked as sector zero.

The next step is to corrupt the test medium a bit (or the RAM image of it if its was tape), to get us into the spirit of things. Experience gained here will pay dividends when you have to use MMSE on a "real" corrupted medium.

Go to the sector editor, Reposition to one of the directory sectors, Edit it (putting in any garbage you want - but take note of whatt you have done) and Commit the sector back. Now experiment with the Rebuild directory and Recover file options. Repeat as necessary. Familiarise yourself with the medium produced by MAKE_TEST_MEDIUM_bas - it will be absolutely identical each time you build it on disk, and virtually identical on tape.

Experiment with corrupting file sectors (you should know where to look for them from the information in the alocation summary and mapping block) in a similar way, using the sector editor. Pay special attention to file headers (corrupt them too - grossly), and reconcile the information in them with what appears in the directory file entry for that file.

Every now and again, choose the menu option of selecting a working device. After reassuring MMSE that you mean what you say, but immediately before you enter the name of the new working device, the existing one will be "released" (ie; no more 'in use' messages wghen attempting access from SuperBASIC). This is a good point at which to CTRL/C into BASIC and adjust the medium, perhaps using MAKE_TEST_MEDIUM_bas or checking files (by copying to the screen) or deleting files (to give you practice with Recover Deleted File - try this "straight", and then try it after you have corrupted the directory). Lastly, experiment after grossly corrupting the map.

7.6 SECTOR EDITOR

This is the most exciting and most powerful part of MMSE: a real workhorse, there is much more here than just an editor. There are all sorts of tools for rapid navigation between sectors, and the collection, concatenation and marking of sectprs. Play about with it by all means; however, when attempting to recover a lost/deleted/corrupted file, first ensure you have exhausted the executive options in the Utilities sub-menu. The sector editor provides "low-level" direct access, and automaticity is much lower than with the Utilities options.

Most cases of problems NOT involving map corruption can be solved in the Utilities section (one exception being recovered deleted files from DISK).

When things HAVE to be solved in the sector editor, it is almost certainly sensible to have obtained a printout of the mapping table (unless you enjoy a lot of troublesome counting working things out from the raw mapping sectors!), the allocation summary and the unfiltered directory.

The sector editor provides an all-encompassing, cocooned environment - you need never leave it (provided you have set up sensible print and output devices) during a session. An array of powerful tools are available to you all at once.

A note - type-ahead has been disabled in some parts of the Sector editor because of the dangers involved if you make an error.

Naming and numbering conventions (all base 0) are as for the utilities described in section 7.5 - you will see more hex here, as hex provides a more compact representation than does decimal (the 3 digit decimal number 255 is just two digits - FF - in hex), which is important as screen space (and printer space - we assume just an 80 column printer) are at a premium. Where decimal is more appropriate, though - as in non sector-content data - we default to decimal.

Abbreviations abound. H stands for Half, Alc for Allocation block and Rlb for relative block.

Let us examine the sector editor screen. The first two lines are self-documenting: the first gives details about MMSE device settings, and the second reports on the file number and name for the sector currently on display. The bottom two lines are the usual key-availability summary and context-sensitive help. These lines will temporarily disappear when certain options are chosen and a dialogue commenced.

Above the bottom two lines is a green band of three lines. This represents the condensed menu system for the sector editor. We'll return to it in a moment.

Immediately above the green band is a status line giving position and ownership information about the sector being viewed. At the right hand end of the line the word UNCOLLECTED will appear if the sector has not been written to a collector file in the current session (a session ends when a new Working device is specified or when you quit from MMSE). We will cover this in more detail soon. Also a BAD SECTOR indicator will appear if the sector is unreadable (whether or not it is marked bad in the map is irrelevant).

Now we come to the main body of the display. The entire sector is displayed, using the character convention described in the last section, 64 characters per line, 8 lines (64*8=512). The numbers in red on the left are to help you count.

Let us look at the green menu options. The cursor - a red slit - is positioned on More. Press SPACE or Enter a few times to confirm that nothing happens. Now move the cursor away with any of the arrow keys (or by typing in the first letter of the menu option you want to go to - if there are a number starting with the same letter, you are cycled around in non-Chinese "reading sequence" each time you press the first letter), experimenting to determine the underlying geometry of the menu screen. Keep each arrow key pressed down in turn and see how fast the cursor flies around, with the context-sensitive help getting updated each time! This is quite relevant, as efficient use of the sector editor often involves rapid use of these options in some cyclic or repeated sequence.

Once you have finished having fun (and with the cursor OFF More) observe that More is written in white while the other options (or almost all the other options if you are on tape) appear in black. This is because More is temporarily disabled. It is disabled for a very simple reason - for this layout, there is no more of the sector to be seen - all 512 bytes are shown. Select Layout and toggle it, and you will see that More is now accessible.

In the new layout, there are two data areas - in character terms (as before) and in hex in the big white-text block. This display is less compact so only 256 bytes of a sector will show at any one time: hence the use of More, which flips between them. The H: (for Half) indicator shows, using a 0 or 1, which bit is currently on display. When you move to a new sector while in the hex layout mode, you are always shown the first half (H:0). Which mode you choose is up to you, but the character mode is more convenient except when you need to know byte values: an up-arrow isn't too helpful in such cases!

Spend a few hours playing with the options in this sector editor menu - the bottom two lines should make things sufficiently clear. The benefit you gain from a lot of hands-on play with MMSE is greater in the sector editor than it is anywhere else. Our comments here are really by exception, where we think clarification is required.

Status info is always full - but unnecessary user dialogue is eliminated by not asking you for things like side number if the disk is single-sided, etc.

You cannot Reposn (Reposition) beyond the physical end of the device (so non-existent sectors cannot be accessed).

Edit works in both display modes - look out for the wrap-around!

Commit is the act of writing the sector (modified by Edit, perhaps) back to the working device, in exactly the same position. This is only likely to be necessary if medium corruption was very minor and very localised. Until you Commit a sector back, the changes to it remain in MMSE workspace (as distinct from the RAM image area in the case of a tape - to modify the RAM image, you must use Commit). If you have just made editing changes to a sector and not committed them, the Collect operation (discussed soon) will, however, collect the modified sector: your modification may have been intended only for the collector file.

Commit will not allow you to 'move' the sector to another physical location on the medium, or to directly change its ownership data (you should do the latter by adjusting the map). The complications that moving sectors around would introduce in terms of the map, directory and block-headers (in the case of tape) makes it an absurdly dangerous thing to do. Patching should be accomplished via a collector file.

If you are desperate enough to want to move the sector elsewhere, try the Klone option which will do it. Use ALTMAP to change the ownership of the current sector. Don't blame us if you miscalculate.

If you Commit or Klone to a sector belonging to the map or directory, and wish the new values to be taken cognisance of immediately, opt for a Reread.

Don't confuse Commit and Collect - they couldn't be more different.

Before using Collect there must be a collector file open - this is opened with Opncoll (and closed with Clscoll - remember to close the file before resetting, or you may be creating for MMSE the job of recovering the recovered data! We will not allow you to open a collector on the working device (you should open the collector on a device that is 100%): ESC and select a more suitable output device (ramdisk is fine): the sector editor is friendly enough to fire up with the same sector that it had when you exited from it.

Collect will write the sector in question to the collector file, whose filename you can select/edit when you choose Opncoll. You cannot have more than one collector file open at a time. You can throw anything into a collector file, but it is probably most helpful to devote one file to each recovered file rather than concatenating the lot in one monolithic collector. Simply toggle the existing collector closed and open a new one with a different name, all without leaving the sector editor.

At the same time, Collect will remove the 'UNCOLLECTED' status flag from the sector.

After a collect, the sector viewed is automatically advanced to the logically next sector on the device (on tape, to the next sector, as logical moves are not defined) - logical sequence having been defined by the physical-logical table: it is the sequence of allocation blocks. We do this to save you time, as far more often than not you will have been moved to the logically-next sector of the file, which is probably where you want to go (to collect this sector too). Repeatedly pressing Space or Enter with the cursor on the Collect key will then provide an easy way of trotting through the medium in logical sequence, picking up sectors as we go along.

Logical sequence will be "broken" if the disk has had an "interesting" history (ie; files have been deleted - overwriting constitutes a deletion and a creation, please note). This is unfortunately common, especially if you are a particularly busy disk user, or just plain miserly with disks. Breaks in logical sequence will cause you to cast about to find the next bit of the file. We'll soon give you some suggestions on how best to do this.

If you opt to Collect a disk sector that is the 'a' sector of an allocation block, MMSE will ask you whether or not you want the first 64 bytes of that sector (in case it is the start of a file, in which case you don't want the header) and whether you want to pick up the next two sectors of the block at the same time.

We suggest you say yes in reply to to this second question to speed things up as WITHIN any allocation block unit, we GUARANTEE the data will be written in logical a-b-c sequence (unless you have used Klone or some other program has stomped on the disk one sector at a time - very very very unlikely). The only disadvantage of this is that you may go a little beyond the logical end of the file - no big deal, you would have done so anyway, unless the file eneded on a sector boundary (why should it?). It probably doesn't matter, but if it does you can always truncate the file either using Editor or by using RESPR/LBYTES/SBYTES (a shorter length) from SuperBASIC. If you opt to have the whole block collected, you are then moved to the start (the 'a' sector) of the next allocation block.

Collect will keep going till you hit the logically last sector/block on the medium.

Note that collect makes no attempt to look at the map for sequencing information. If there are valid entries in the map for that file, you probably shouldn't be bothering to collect it but instead just using COPY from SuperBASIC or a toolkit WCOPY. We realise that the medium may have a partly-corrupted map or corrupted directory, and the bit of the map containing references to this file were OK - but its too remote a case. Logical sequence is fine - most of the time. Keep an eye on sector contents to ensure you haven't "strayed" (the file-deletion problem). The map-derived info on File number, File name and Rlb (Relative block within the file) will be presented on-screen in any event - keep an eye on them, if you prefer.

If and when the logical move (performed automatically after the collect) gets you out of the as-yet-not-fully-collected file, this is how best to pick up the threads. Don't collect a 'wrong' sector or block. Use the menu option for a Logical move left and you will be back to the last sector of the file.

Have a look around and decide what the next sector is likely to have at the beginning of it (eassy if it is a text file, source listing or recognisable data (like Archive/Quill stuff). Look up the allocation summary and hunt for it by looking at the first 64 byte summary (as implied earlier, the trauma will only occur at block boundaries and not within blocks).

It is quicker to take a peek at the mapping table first - if it is relatively uncorrupted - look to the screen for the (map-derived) file number and relative block number for the last sector of the desired file found so far. Say it is file 13, relative block 4. Look up the mapping table to see which sector contains the next relative block of the file (look for $00D/005 - or $0D/05 if it is on tape), and Reposn yourself to it.

If these suggestions fail, Reposn to the first physical sector (Tr:0 Sd:0 Sc:0 on disk, Sector 0 on tape) and do a Search (it will be case insensitive) for a string that you know will occur in the next few blocks of the file and in not too many other places. Even if the file was largely unprintable or in code, there should be some ASCII strings in it - check old versions or printouts of the file for ideas, or its documentation if it is commercial stuff. Search goes in physical sequence through the medium (on disk, cycling through the 9 sectors (0...8), then flipping side if appropriate, flipping through the 9 sectors, flipping side again and skipping to the next higher track track. If a wrong match is found, a further Search will resume progress from that point. Even if you find a segment of the desired file a little way logically beyond the immediately next missing sector, it does not matter - a logical backwards move (2nd line of green menu, fifth option) should get you there.

If you can't find a suitable search string either, you may have to wander around the medium till you hit the right data. This is made less time-hungry by the Uncollected move commands.

Let us review all the move (ie; sector navigation) commands available. We won't classify Collect (or Bulkwrt, which is related to collect and which we shall soon meet) as a move command, since it has other executive effect too. Also, lets rule out Reposn, which isn't a single keypress command and is hence too slow for anything but long-distance moves to known, definite destinations.

All the other sector navigation commands are abbreviated to a move type followed by a left (signifying previous) or right (signifying next) arrow. All move commands (like Recover) will allow selection of sectors marked bad; deleted or unallocated - but not ones that are non-existent.

More importantly, all will allow multiple entries with type-ahead. A lazy screen feature is implemented, we won't slow down the move by repainting the screen if there are more commands on the way in the keyboard buffer!

We have already encountered the Logical moves (Collect uses a Next logical).

File moves assume a valid map and read data from the map. You move through the file (user files, directory or map) in logical sequence, and you can't move off at either end. Attempt at file movement when the sector belongs to other special file types will restrict your movement to the block you are in - after all, there is no sequence in the deleted/unallocated/bad pools!

Physical moves are obvious - they follow the same sequence as did Search.

Uncollected moves involve movement to the next uncollected sector, in physical move sequence. This is useful, as hinted at earlier, as your attention is focussed solely on uncollected sectors. Combined use of Next/Previous Uncollected and Collect save a lot of time. Develop (say) a U U SPACE C C C C SPACE rhythm to race through the medium.

There are three more commands to help you here. Hide removes the 'Uncollected' flag from a sector - sectors you encounter on your travels dealing with the fragmented/bad-map/non-ASCII-file-filled medium that have nothing to do with the file you are recovering (or with other files you are likely to want to recover in the same session) can be hidden. This means that though they have not actually been collected they will be classified as collected, and Uncollect will skip over them.

Show will unhide a sector. Use show to correct mistakes made when using Hide, or to make an already collected sector reachable by an Uncollected move.

Flags is a fascinating option. You will get an overview of all the sectors on the medium (on a disk, presented in batches of 18, 4 batches per line, 20 lines, 18 * 4 * 20 = 1440). U denotes a sector is uncollected, . that it is collected, and @ gives the current position.

By narrowing down the field of uncollected sectors (using Collect and Hide), the Uncollected moves will allow rapid scans of the medium.

Let us summarise the moves:

	CONSULTS MAP	A-B-C SEQUENCE	REQUIRES
Physical	No	No	-
File	Yes	Yes	Good map
Logical	No	Yes	Clean history desirable
Uncollected	No	No	Intelligent use of Hide

Similar to Search is Locate, except that it does a case-insensitive search only within the area where a filename could be present (ie; in file headers). You are allowed to specify whether the match is Beginning (matches with start of filename), Instring (matches any part of filename) or Complete (matches filename exactly). To find the start of a file which you want to collect on a disk with a bad map, a Locate may find it faster than you will scanning the allocation summary.

Let us now consider Bulkwrt. This is a sort of global collect, where it first Locates the start of the file for you, then does a number of automatic collects. You tell it the length of the file being recovered - it will work out how many collects to do. Nifty, eh? It is worth trying even on disks with chequered histories, as if it works, it will have saved an awful lot of keypresses....

Calc is best experimented with yourself.

Please use the sector editor for practice on disks generated by the MAKE_TEST_MEDIUM_bas program, including ones which you corrupt. This is the only way to master the Sector editor...

7.7 MS-DOS/TOS FILE COPIER

When you choose this option an ancillary program module, MMSE_XOVER, is loaded from the system device.

This sub-menu enables you to use disks formatted under MS-DOS and Atari-TOS on your Sinclair QL with disk drive. All standard file operations are available. It also includes a facility to translate text files between the different character sets of QDOS, MS-DOS & TOS.

The disk interface attached to your QL must conform to the QL floppy disk standard. This is necessary because MMSE_XOVER accesses the floppy disk sectors directly to handle the alien disk formats. The device name of the disk drive has to be either 'flp' or 'fdk'.

The Atari TOS disk format is almost identical to the MS-DOS format. There are only minor differences, which are recognised and taken into consideration by MMSE_XOVER automatically, so that you need not explicitly specify what disk you are using. In the following description of the program's functions the term 'DOS' always includes the Atari TOS format.

7.7.1 DISPLAY DISK INFORMATION

You are asked for the drive name which contains the disk to be examined (e.g. flp1_). A default name is displayed, which can be accepted by pressing ENTER or altered with the usual line-editing facilities.

Depending on the disk format that is recognised, one of the following information screens appears:

```
disk type ..... QDOS
medium name ..... < name of disk >
random number ..... < random value for identification >
update count ..... < write accesses since FORMAT >
free sectors ..... < free sectors >
good sectors ..... < usable sectors >
total sectors ..... < total sectors >
sectors/track ..... < sectors per track >
sectors/cylinder .... < sectors per cylinder >
number of tracks .... < number of tracks >

disk type ..... DOS/TOS
OEM name ..... < manufacturer identification >
version ..... < DOS version or TOS random value >
media bytes ..... < format identification (see below)>
sides ..... < sides of disk >
sectors per track ... < sectors per track>
total sectors ..... < total sectors >
reserved sectors .... < reserved sectors >
FATs ..... < number of file allocation tables >
sectors ..... < sectors per FAT >
max. dir entries .... < max. number of files in root
                       directory >
```

It is not necessary to understand all of the above information to use the MS-DOS/TOS File sub-menu. The main purpose of this function is to distinguish between QDOS- and DOS-formatted disks. Additional information on the disk formats is contained in the documentation to the QL floppy-disk standard and in various books on MS-DOS and on the Atari ST.

7.7.2 SHOW QDOS OR DOS DIRECTORY

When you have entered the drive name, MMSE_XOVER automatically recognises the disk format and shows the directory. In addition to the file names, the length and the date stamp are displayed. When the output window has filled, the program waits for a key to be pressed. Then the window is scrolled up and another page of files is listed.

The directory function is not limited to disk drives. It may be used to list the files on all other QDOS directory devices (like RAM disks) as well.

If the root directory of a DOS disk contains subdirectories, these entries are displayed and are marked with '<dir>'. MMSE_XOVER does not support subdirectories (rare on floppies) except for giving their creation date.

7.7.3 IMPORT FILES (DOS TO QDOS)

This function enables you to transfer files from DOS to QDOS. On the QDOS side the files need not to be on a disk drive; it is possible to transfer files to RAM disk or microdrive.

When you have entered the source and the destination device, you are asked: 'go through directory? (Y/N)'.

Replying with 'y' will cause MMSE_XOVER to work through the list of files in the directory of the source device. For each file you are asked whether a transfer is to be performed. The selection is made in a similar way as with the wildcard commands of many toolkits: '(Y/N/A/Q)' is displayed, and pressing

'y' transfers the file;
'n' does not transfer the file;
'a' transfers this and all following files;
'q' stops the selection facility.

Within this selection facility the name of the destination file is generated from the name of the source file. As a DOS filename consists of only 8 characters (+ 3 characters extension), any additional characters are omitted. Occasionally this may lead to problems when long QDOS filenames differ from each other only in the last characters. Cutting the last characters then leads to identical names, so that the filename already exists on the DOS disk.

If you choose to spurn the selection facility described above by typing 'n', you may enter the source and destination file names explicitly. Name and extension in a DOS filename may be separated by a '.'. The formatted filename is then echoed to the screen.

7.7.4 EXPORT FILES (QDOS TO DOS)

This function enables you to transfer files from QDOS to DOS. On the QDOS side the files need not to be on a disk drive; it is possible to transfer files from RAM disk or microdrive. The procedure for using this option is much the same as for the inverse transfer option described in Sub-Section 7.7.3 .

7.7.5 DELETE DOS FILES

As with the file transfer functions, it is possible to work through the specified directory file by file or to specify the files individually.

7.7.6 RENAME DOS FILE

As with the file transfer functions, it is possible to work through the specified directory file by file or to specify the files individually.

7.7.7 FORMAT ALIEN MEDIUM

This function generates a DOS- or TOS-formatted disk by writing the system sectors of the selected format to a QL-formatted disk. The following formats are available:

F1 ... PCDOS 360k PC/XT (DS/40 track/9 sectors)
F2 ... PCDOS 720k PS/2 (DS/80 track/9 sectors)
F3 ... Atari TOS single-sided
F4 ... Atari TOS double-sided

Important: MMSE_XOVER writes only the system sectors of the specified format to the disk. Therefore it is necessary to pre-format the disk with the QDOS command FORMAT - with the needed number of tracks and sides. The command to set these parameters differs from one disk controller to the other.

The advantage is that the operation is much faster on an already formatted disk. The disadvantage is that you have to make sure that there are no defective sectors on the disk, i.e. the message from the QDOS command FORMAT 'x / y sectors' must show x and y to be equal!

7.7.8 CONVERT QDOS TEXT FILES

Though QDOS, MS-DOS and TOS all use the ASCII character set, the special characters (Codes > 128) are totally different between QDOS and DOS/TOS. Even between DOS and TOS there are minor differences.

The convert function performs the best achievable conversion between character sets. A complete conversion cannot be performed because some characters do not exist in all three character sets.

In addition to the character conversion, the end-of-line code is changed. QDOS uses only a CR character, while DOS/TOS needs a CR-LF sequence.

The conversion facility works only on files which are on a QDOS device. So transfer the file to QDOS first, and then convert it!

7.7.9 RETURN TO MAIN MENU

You guessed it! This will leave MMSE_XOVER sub-menu and return you to the main menu - you may need to press CTRL/C to re-awaken MMSE. Note that MMSE has been sleeping in RAM all along - it does not need to be reloaded or for the system to be given any access to the medium containing the MMSE file.