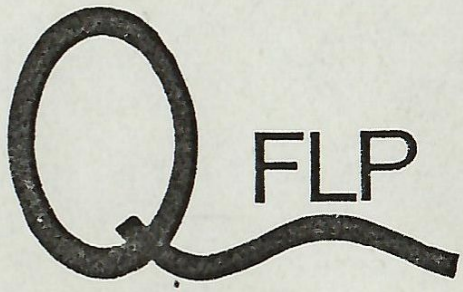



# Floppy Disk Driver

# For the Sinclair QL

# QFLP Disk Upgrade for MicroPeripherals Disk Systems

## Versions

The MicroPeripherals QL Disk Interface was supplied in at least two different versions. The main versions were version 3 (usually identified the startup banner as V3.3) and version 5 (identified as V5.3). QFLP EPROMs are identified as V1.193 (for version 3) and V1.19 (for version 5).

## Installation Instructions

### Removing the MicroPeripherals EPROM

Note the position of the notch on the short edge of the chip. Prise the chip gently out of its socket with a screwdriver or IC extraction tool, if you have one, being careful to apply even pressure so as not to bend any of the metal legs.

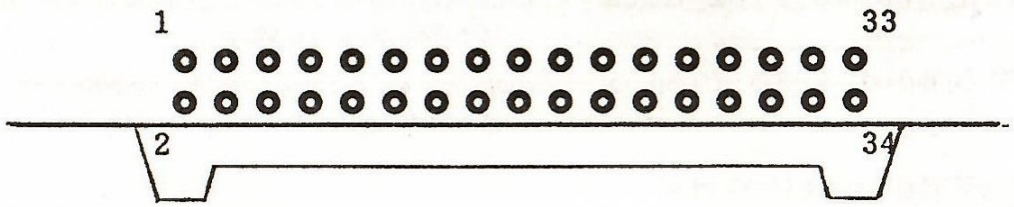
### Inserting the QFLP EPROM

Ensure that the legs on the new EPROM are straight and are at right angles to the chip. If the pins straddle the socket holes it may be necessary to correct the position of the legs by pressing them against a hard, flat surface. Hold the chip with the notch in the same position as the Micro Peripherals one was, locate the EPROM over the socket and then press it firmly into place.

To obtain maximum performance from your upgrade you may remove the cover from the disk interface and put a shorting link between pins 33 and 34 on the floppy disk connector. Pins 33 and 34 are the furthest from the arrow on the connector. If you ever wish to replace the MicroPeripherals EPROM, this shorting link must be removed.

Underside view of MicroPeripherals Disk interface

(Standard pin numbering)




The READY line is on pin 34 and its earth return is on pin 33. For best performance, these pins should be connected together.

## WARNING

The information for an EXECutable file is stored in a non-standard way by the MicroPeripherals interface. This means that any EXECutable files should be transferred to Microdrives before the EPROM is changed. They may then be copied back to disk after the QFLP ROM has been installed. Note that the free sector counts will be incorrect for disks recorded using the Microperipherals EPROM.

## The Floppy Disk Driver

The QL computer is delivered with two "mass storage" devices: the Microdrives. These devices have the same function as the floppy disks on more expensive personal computers, being designed for the permanent storage of programs and data. Other devices which behave in the same way as Microdrives (such as floppy or hard disks) may be added to the QL "transparently". This means that QDOS will ensure that a program does not need to "know" where its data is stored. A Microdrive looks, to a program, exactly the same as a floppy disk. This "device independence" is a built-in characteristic of the QDOS operating system.

The simplest way of using a floppy disk system on the QL is to copy all programs and data to floppy disks, and either add the command "FLP_USE MDV" to all BOOT files, or type this command at the start of a session on the QL. The effect of this command is to make the floppy disks pretend to be rather large and fast Microdrives.

For example, a modified BOOT file for executing the PSION program Quill could look like:

```
90 FLP_USE mdv:      REMark - emulate microdrives
100 CLOSE #1: CLOSE #2
110 EXEC_W mdv1_quill
....
....
```

On the other hand, it is just as easy to use the floppy disks without changing the name. All the filing system commands described in the "Microdrives" section of the QL Concept Reference Guide will work with floppy disks, provided the filenames start with "FLP" instead of "MDV":

| | |
|---|---|
| `FORMAT flp1_test` | *formats a new floppy disk in drive 1* |
| `DIR flp1_` | *directory listing of floppy disk 1* |
| `SAVE flp1_myprog` | *save the current SuperBASIC program as "myprog" in floppy disk 1* |
| `OPEN_NEW #3,flp2_data` | *creates and opens a new file "data" in floppy disk 2* |
| `COPY mdv1_x TO flp1_x` | *copies file x from Microdrive 1 to floppy disk 1* |

## Floppy Disk Compatibility

The QJUMP Floppy Disk driver not only provides all the built-in Microdrive filing system operations, but includes the extended filing system operations provided in the Sinclair QL Toolkit and QJUMP Super Toolkit II for Microdrives. This allows all the SuperBASIC extensions provided in the Toolkits (e.g. FOP_OVER, RENAME etc.) to be used with the floppy disks

OPEN OVERWRITE  Trap #2, D0=1, D3=3

> This variant of the OPEN call opens a file for write/read whether it exists or not. The file is truncated to zero length before use.

RENAME Trap #3, D0=4A, A1 points to new name

This call renames a file. The name should include the drive name (e.g. FLP1_NEW_NAME).

TRUNCATE Trap #3, D0=4B

This call truncates a file to the current byte position.

In addition the FS.FLUSH call for a file, not only flushes all the file buffers, but, unlike the Microdrive driver, updates the map and the directory. This means that a new file can be created, and if it is flushed, then in the event of the QL being turned off or reset before the file is closed, then all of the file (up to the point where it was last flushed), is readable. In effect a FLUSH call is just the same as a CLOSE call, except that the file remains open and the file pointer remains unchanged.

## Auto-boot

If there is a disk in drive 1 when the QL is turned on (this may be risky with some makes of floppy disk drive, particularly those with permanently loaded heads) or reset (this should be safe with all drives), then the QL will boot from the disk in drive 1, otherwise the QL will boot from Microdrive 1 as usual. There is no direct control over the Disk drive motor, the motor is turned off by the hardware in the face after 10 disk rotations. To stop the motor, insert a disk into drive 1.

When a "directory device", such as a floppy disk, is accessed for the first time, QDOS will allocate a block of memory for the device. In the case of a floppy disk, the Sinclair standard format requires a block of memory about 1.6 kilobytes long. This is rather larger than the Microdrive block which is only about 0.6 kilobytes long. The auto-boot procedure used ensures that if there is no disk in drive 1 when the QL is reset, then the 1.6 kilobyte block for disk drive 1 will not be allocated. Programs that are too large to execute when floppy disks are being used, should still execute from microdrives.

## Microdrive Emulation

The standard drivers also includes the SuperBASIC procedure FLP_USE to change the name of the floppy disk driver.

```
FLP_USE mdv or FLP_USE 'mdv'
```

reset the name of the floppy disk driver to "mdv", so that all subsequent open calls for Microdrives will use the floppy disks instead.

For Example

```
FLP_USE mdv
....
....
OPEN #3,mdv1_myfile
```

will actually open the file "myfile" on floppy disk 1, rather than trying to open a file on Microdrive 1.

Any three letters may be used as a new device name, in particular the driver can be reset by the command:

```
FLP_USE flp
```

## Floppy Disk Options

There are three parameters of the floppy disk system which are available as user options.

The security level is selectable to allow a user to choose higher speed of access at the cost of reduced immunity to erroneous disk swapping. There are three levels of security, the lowest level still being at least as secure as common disk based operating systems (e.g. MSDOS and CPM).

A user may specify the time taken for the disk drive motor to get the disk speed to within the specification.

A user may specify the number of tracks to be formatted on a disk.

The parameters are specified by three forms of the FLP_OPT command:

FLP_OPT *security level*
FLP_OPT *security level, start up time* (in 50ths of a second)
FLP_OPT *security level, start up time, number of tracks*

## Security

The Microdrive filing system is unusual in that, although the data is stored in "sectors" in just the same way as on a floppy disk, each sector holds information which identifies the cartridge. When a cartridge is changed the filing system will recognise the change the next time any access is made to Microdrive. Standard floppy disk formats do not allow this type of security, so the format used for QL floppy disks includes identifying information in Track 0 Sector 1 of the disk. Clearly if this were checked every time any access were made to the disk, then the floppy disk system would be very slow indeed. Security, in the context of this user option, is the extent to which the floppy disk system may be abused by changing disks, while they are in use, without destroying data stored on the disks.

There are four operations which affect the security: the first is the operation to check if the disk has been changed, the second is the operation to flush the slave blocks, the third is the operation to update the map and the fourth is the operation to update the directory.

In these definitions, the term "the drive has stopped" is usually taken to mean that the motors have stopped and no drive select light is visible.

## Security Level 0

The disk is only checked when a file is opened and the drive has stopped since the last time it was checked and there are no files already open on the drive. The map is only updated after a file is closed (or flushed) when half a second has elapsed without any other disk operation.

At this lowest level of security, confusion or loss of data can be expected if a disk is changed while there are still files open or the motor is running.

### Security Level 1

The disk is checked when a file is opened, or data or the map is to be written, and the drive has stopped since the last time it was checked. The map is only updated after a file is closed (or flushed) when half a second has elapsed since the previous disk operation.

At this level of security, disks should only be changed while the motor is stopped (all select lights off). If a disk is changed while there are files open, then read operations will be confused but any write operations will be aborted. This should maintain the integrity of the data on the disk.

### Security Level 2

The disk is checked whenever a file is opened or whenever the map or data is to be read from or written to the disk and the drive has stopped since the last time the disk was checked.

The map and directory are updated and the buffers are flushed immediately after a file is closed, or after an FS.FLUSH call.

This is the default security level and data should be quite secure unless a disk is changed while the motors are running.

### Security System Errors

There are two error messages which may be written to the screen by the floppy disk filing system. These are in the form of the disk name followed by the message itself. The first message indicates that an attempt to read or write a sector on the disk has failed:

*disk name* read/write failed

The second message indicates that a disk has been changed while it is still in use:

*disk name* files still open

If the floppy disk system attempts to write to a disk which has been changed, then you may get both messages indicating that the attempt to write the data has been aborted, and that files were still open when the disk was changed.

### Start Up Time

The floppy disk system will always try to read data from a disk as soon as it can. However, to preserve the data integrity of the disk, write operations are held up until the disk has been "run up" for long enough for the speed to be stable. As a default this is set to .6 second which is more than enough for most modern drives. The start_up_time parameter is in 20 millisecond units, so the default value is 30. A value of 13 (260 milliseconds) is adequate for the most recent direct drive 3.5 inch drives, while some older drives may require a value of about 60 (1.2 seconds).

## Number Of Tracks

The QL format for disks allows the number of tracks on a disk to be read from the disk itself. However, the number of tracks must be determined when a disk is to be formatted. Normally the disk system will do this itself by checking if there are at least 55 tracks on a disk. If there are, then there are assumed to be 80 tracks, otherwise it is assumed that there are 40 tracks. This internal check may be overwritten, allowing 37 track and 75 track drives to be formatted as well as saving possible wear or damage to a 40 track drive when seeking track 55 (somewhere in the middle of the jacket).

## Direct Sector Read/Write

The software includes provision for reading sectors of a disk using direct addressing. To do this a special file is opened on the disk. The name is

| | | |
|---|---|---|
| FLPn_*Dsd | *where s is the sector length* | *0=128 bytes*<br>*1=256 bytes*<br>*2=512 bytes*<br>*3=1024 bytes* |
| | *and d is the density* | *D=double (MFM)* |

When opening a disk for direct sector read/write from SuperBASIC, the name should be enclosed in quotes (or apostrophes).

```
OPEN #3,'flp1_*d2d'
```

When this file is open, no other file may be open on the drive. The only IO calls supported for this type of file are IO.FSTRG, IO.SSTRG, IO.POSAB and IO.POSRE, to read or write complete sectors or to set the position. The parameter (D1) to the POSRE call is ignored, but the current position is returned. Reading or writing a sector does not change the file position.

The position is a composite of the required sector, side and track:

```
sector number + side * 256 + track * 65536
```

To ensure compatibility with string IO the length specified in the SSTRG and FSTRG calls may be one of three values:

| | |
|---|---|
| sector length | the complete sector is read or written |
| 2 | returns the sector length (IO.FSTRG)<br>ignored (IO.SSTRG) |
| 2 + sector length | returns the sector length followed by the sector (IO.FSTRG)<br>skips the first two bytes, and writes the rest to the sector (IO.SSTRG) |

This variety enables sectors to be read and written in SuperBASIC using the normal string IO in the Super Toolkit II, as well as by assembler programs. For example, sector 1 of side 1 on track 2 may be read into the string A$ using the following command:

```
GET #n\1+256+2*65536, a$
```

When using the direct sector read/write calls for a 40 track disc in an 80 track drive, the track number should be doubled. Seek errors will not be detected. If a read/write error is returned from a direct sector read/write call, then it will be safest to make another call to read from track zero. Calls to read from or write to track zero will cause a "restore" rather than a seek, and will thus reset the drive to a known state.

## Disk Drive Specifications

It is a requirement that disk drives used with this version of the disk driver should be set to have the motor on when provided with a "motor on" signal and there is a disk in the drive. Drives which turn the motor off when the drive is not selected will not give reliable service.

The disk driver will automatically adjust itself to use any mixture of disk drives, 40 or 80 track, single or double sided. In addition it will adjust itself to use slow step rate drives. Disks need not have been formatted and written on the same specification drive as a drive being used to read them.

### Compatibility chart

| Disk format -> | 40T SS | 40T DS | 80T SS | 80T DS |
|---|---|---|---|---|
| Drive | | | | |
| 40T SS | C | ? | X | X |
| 40T DS | C | C | X | X |
| 80T SS | R | ? | C | ? |
| 80T DS | R | R | C | C |

C = compatible
R = compatible (read only)
X = incompatible
? = incompatible but may not be detected correctly on some types of drive

The format procedure automatically checks the drive specification and will format the drive in an appropriate manner. Note that 40 track drives which do not have an end stop, or which would suffer damage when stepped beyond the 40th track (to track 55) should not be formatted unless the number of tracks has been specified in an FLP_TRACK command. It is possible to force the disk driver to format a disk as single sided on a double sided drive by making the 11th character (it is invisible) of the medium name an asterisk: e.g.

```
FORMAT 'FLP1_DISK_NAME *'
```

## QFLP Extensions to SuperBASIC

The extensions to SuperBASIC incorporated in the QFLP upgrade are all concerned with the QL filing system. These extensions are not available immediately after resetting the QL to avoid conflicts between the names of these extensions and any you might use in your own programs. They are invoked by the command:

```
FLP_EXT          (no parameters)
```

This command may be included in a BOOT file or typed on the keyboard at any time. If you have AH or JM ROMs in your QL, you will not be able to use the extensions in the same boot file as the FLP_EXT command. In this case LRUN a second boot file from the first.

```
EXTRAS
EXTRAS #channel
```

will list the extra procedures and functions linked in.

## DATA_USE - Data File Default

Default directories may be set for use with many Toolkit commands.

```
DATA_USE directory name
```

If the directory_name supplied does not end with '_', '_' will be appended to directory name. The directory name can be more detailed than just a device name. For example:

```
DATA_USE flp1_project5_library
....
WDIR
ferr=FOP_NEW (#3,fred)
```

WDIR will produce a directory listing of all filenames starting with 'flp1_project5_library' and then FOP_NEW will open a new file called 'flp1_project5_library_fred'. The default set by this command is optional and is only used if the name supplied to a Toolkit command is not a valid file or device name. Thus:

```
ferr=FOP_NEW (#3,flp2_fred)
```

will open file 'flp2_fred' (not 'flp1_project5_library_flp2_fred'!)

## File Open Functions - FOPEN
## FOP_IN FOP_NEW FOP_OVER FOP_DIR

This is a set of functions for opening files. These functions differ from the OPEN procedures in ROM in two ways: firstly, if a file system error occurs (e.g. 'not found' or 'already exists') these functions return the error code and continue; secondly the functions use the DATA_USE directory default

```
FOPEN (#3,name)        open for read/write
FOP_IN (#3,name)       open for read only
FOP_NEW (#3,name)      open a new file
FOP_OVER (#3,name)     open a new file, or overwrite old file
FOP_DIR (#3,name)      open a directory
```

Directory entries may be read using GET to get information. Each entry is 64 bytes long, the length of the file is at the start of the entry, there is a standard string starting at the fourteenth byte of the entry giving the filename and there is the update date as a long integer starting at the fifty sixth byte.

### Example of File Open

A file may be opened for read only with an optional extension using the following code

```
ferr=FOP_IN (#3,name$&'_ASM')        :REMark open _ASM
IF ferr=-7: ferr=FOP_IN (#3,name$)   :REMark try no _ASM
```

### File Enquiry - FLEN FTYP FDAT

There are three functions to extract information from the header of a file. Note that in current versions of the Microdrive handler, the header is only updated on an FS.HEADS call or on closing the file. The QJUMP Floppy Disk Driver also updates the header on a call to flush the disk buffers. This means that the file length read from the header will usually be the file length as it was when the file was opened. If a file is being extended, the file length can be found by using the FPOS function to find the current file position. (If necessary the file pointer can be set to the end of file by the command GET #n\999999.)

| | |
|---|---|
| FLEN (#*channel*) | *returns the file length* |
| FTYP (#*channel*) | *returns the file type* |
| FDAT (#*channel*) | *returns the data space of EXEC files* |

### BGET BPUT GET PUT FPOS - Direct IO

In QDOS, files appear as a continuous stream of bytes. On directory devices (Microdrives, hard disks etc.) the file pointer can be set to any position in a file. This provides 'direct access' to any data stored in the file. Access implies both read access and, if the file is not open for read only (OPEN_IN from SuperBASIC, IO.SHARE in QDOS), write access. Parts of a file as small as a byte may be read from, or written to any position within a file. QDOS does not impose any fixed record structures upon files: applications may provide these if they wish.

Procedures are provided for accessing single bytes, integers, floating point numbers and strings. There is also a function for finding the current file position.

The general form of the direct I/O commands is:

*command #channel, item, item ....*
*or command #channel\pointer, item, item ....*

It is usual (although not essential - the default is #1) to give a channel number for the direct I/O commands. If the pointer is given, the file position is set before processing the list of I/O items; if the pointer is a floating point variable rather than an expression, then, when all items have been read from or written to the file, the pointer is updated to the current file position.

## Byte I/O

```
BGET #channel, list of items
BGET #channel\pointer, list of items
BPUT #channel, list of items
BPUT #channel\pointer, list of items
```

BGET gets 0 or more bytes from the channel. BPUT puts 0 or more bytes into the channel. For BGET, each item must be floating point or integer variable; for each variable, a byte is fetched from the channel. For BPUT, each item must evaluate to an integer between 0 and 255; for each item a byte is sent to the output channel.

For example the statements

```
abcd=2.6
zz%=243
BPUT #3,abcd+1,'12',zz%
```

will put the byte values 4, 12 and 243 after the current file position.

## Unformatted I/O

It is possible to put or get values in their internal form. The PRINT INPUT commands of SuperBASIC handle formatted IO, whereas the direct I/O routines GET and PUT handle unformatted I/O. For example, if the value 1.5 is PRINTed the byte values 49 ('1'), 46 ('.') and 53 ('5') are sent to the output channel. Internally, however, the number 1.5 is represented by 6 bytes (as are all other floating point numbers). These six bytes have the value 08 01 60 00 00 00 (in hexadecimal). If the value is PUT, these 6 bytes are sent to the output channel.

The internal form of an integer is 2 bytes (most significant byte first). The internal form of a floating point number is a 2 byte exponent to base 2 (offset by hex 81F), followed by a 4 byte mantissa, normalised so that the most significant bits (bits 31 and 30) are different. The internal form of a string is a 2 byte positive integer, holding the number of characters in the string, followed by the characters.

```
GET #channel, list of items
GET #channel\pointer, list of items
PUT #channel, list of items
PUT #channel\pointer, list of items
```

GET gets data in internal format from the channel. PUT puts data in internal format into the channel. For GET, each item must be a integer, floating point, or string variable. Each item should match the type of the next data item from the channel. For PUT, the type of data, put into the channel, is the type of the item in the parameter list. The commands

```
fpoint=54
...
wally%=42: salary=78000: name$='Smith'
PUT #3\fpoint, wally%, salary, name$
```

will position the file, open on #3, to the 54th byte, and put 2 bytes (integer 42), 6 bytes (floating point 78000), 2 bytes (integer 5) and the 5 characters 'Smith'. Fpoint will be set to 69 (54+2+6+2+5).

For variables or array elements the type is self evident, while for expressions there are some tricks which can be used to force the type:

```
....  +0      will force floating point type
....  &''     will force string type
....  ||0     will force integer type
```

```
xyz$='ab258.z'
...
BPUT #3\37,xyz$(3 to 5)||0
```

will position the file opened on channel #3 to the 37th byte and then will put the integer 258 on the file in the form of 2 bytes (value 1 and 2, i.e. 1*256+2).

Provided no attempt is made to set a file position, the direct I/O routines can be used to send unformatted data to devices which are not part of the file system. If, for example, a channel is opened to an Epson compatible printer (channel #3) then the printer may put into condensed underline mode by either

```
    BPUT #3,15,27,45,1
or  PRINT #3,CHR$(15);CHR$(27);'-';CHR$(1);
```

Which is easier?

## File Position

There is one function to assist in direct access I/O: FPOS returns the current file position for a channel. The syntax is:

```
FPOS (#channel)
```

For example:

```
PUT #4\102,value1,value2
ptr = FPOS (#4)
```

will set 'ptr' to 114 (=102+6+6).

The file pointer can be set by using any of GET, BGET, PUT or BPUT with no items to be got or put. If an attempt is made to put the file pointer beyond the end of file, the file pointer will be set to the end of file and no error will be returned. Note that setting the file pointer does not mean that the required part of the file is actually in a buffer, but that the required part of the file is being fetched. In this way, it is possible for an application to control pre-fetch of parts of a file where the device driver is capable of pre-fetching.

## VIEW - Examining a File

VIEW is procedure intended to allow a file to be examined in a window on the QL display.

```
VIEW name                 view a file: lines are truncated to
VIEW #channel,name        fit in the window and, when the
                            window is full, CTRL F5 is generated.
```

## RENAME and TRUNCATE

The RENAME and TRUNCATE procedures operate on files on floppy disks or on microdrives if the Sinclair QL Toolkit has been added. If either of these procedures are used on a standard QL to operate on a microdrive file, the result will be a 'bad parameter'.

| | |
|---|---|
| RENAME *old,new* | *rename a file, the DATA_USE default directory is used for both filenames.* |
| TRUNCATE *#channel* | *truncate the file open on #channel to the current file position.* |

## Wild Card Commands

There is a set of directory maintenance commands using a 'wild card' definition of the file name (based on the DATA_USE default directory).

| | |
|---|---|
| STAT *name* | *print medium name, number of free sectors, total number of sectors.* |
| WDIR *name* | *list directory, generates CTRL F5 when the window is full.* |
| WSTAT *name* | *list file name, length and last update date, generates CTRL F5 when the window is full.* |
| WDEL name | *delete files (requests confirmation).* |
| WDEL_F name | *delete files (forced).* |

Each of these commands (except WDEL_F) may be used with a channel number to send the output to a particular window or file. When using WDEL, each filename is written to the chosen channel, and the user is requested to press one of the keys:

| | | |
|---|---|---|
| Y | (yes) | *delete this file* |
| N | (no) | *do not delete this file* |
| Q | (quit) | *do not delete this or any of the next files* |
| A | (all) | *delete this and all the next matching files* |

The name in these procedures may refer to more than one file. To do this file names are divided into sections (flp2_fred_bin has three sections) and a name may have missing sections (e.g. flp2_old__list has one missing section). All those files whose names have sections matching the sections in the given name are referenced by the commands. In the following examples, flp2_ is assumed to be the default data directory.

| Wild name | Typical matching files |
|---|---|
| fred | flp2_fred<br>flp2_freda_list |
| _fred | flp2_fred<br>flp2_freda_list<br>flp2_old_fred<br>flp2_old_freda_list |
| flp1_old__list | flp1_old_jo_list<br>flp1_old_freda_list |