PCML Q+ DISK INTERFACE FOR THE SINCLAIR QL COMPUTER

(INCLUDING THE PCML TOOLKIT V 1.14/ V 1.16)

CONTENTS

PCML Q+ DISK INTERFACE for the Sinclair QL computer.

INTRODUCTION

Thankyou for buying the PCML Q+ Disk Interface. It gives you the facility to have up to 2.88 megabytes of disk storage capacity, and the ability to set up Ram disk's. The disk interface can also be upgraded to include a 256 K memory expansion.

Installation of the Q+ Disk Interface.

1. Disconnect the QL from the mains.
2. Remove the plastic cover from the left-hand side of the QL. It is often a very tight fit, and may require some effort.
3. Insert the board side of the disk interface into the QL (component side up).
4. Push the disk interface firmly in until the plastic case fits flush with the QL case.

Fitting the Disk Drives to the Interface.

1. Disconnect the QL and the disk drives from the mains.
2. Connect the plug from the disk drives to the Q+ disk interface (the connector is polarised to prevent confusion, and is a standard Shugart connection).

Testing of the Q+ interface.

Power up the QL and the disk drives at the same time (without disks in the drives).
If the interface has been expanded with 256 K of memory, there will be a longer delay than normal, while this memory is tested. If the interface is functioning correctly, a sign on message will now be displayed,

```
          PCML Disk/RAM System  V1.14  c 1984
```

If you then press F1, or F2, the QL will attempt to boot from FLP1 instead of MDV1.

The PCML TOOLKIT software provides a means of instant access data storage using the standard I/O system calls. It not only provides all the built-in microdrive filing system operations, but also the microdrive extensions provided in the Sinclair QL Toolkit.

EXTENSIONS TO SUPERBASIC

FORMAT

FORMAT can now format "ram-disks", and floppy disks, as well as microdrives.

```
syntax          FORMAT device_(optional extension)
```

Device ,in the above command, can now be MDV1, FLP1, or RAM1 depending on the device to be formated. In the case of microdrives or floppy disks, the optional extension would be the name of the cartridge, or disk, eg 'newdisk' or 'data'. In the case of the ram-disk, the extension is the size of the ram-disk required, in sectors of 512 bytes, ie, the same format as the microdrives, and is not optional. A ram-disk can be cleared by reformating, or completely removed by ignoring the extension or using an extension of 0.

It is possible to format up to 8 ram-disks (1 to 8) as long as there is sufficient memory available.

USE

```
syntax          (alternative device)_USE MDV
                (alternative device)_USE 'MDV'  (the space after
                                                 USE is optional)
```

The USE command can be be used, primarily, to redirect all the microdrive commands so that all subsequent calls for a microdrive use the alternative devices.

eg, RAM_USE MDV will divert calls to MDV1 to RAM1, MDV2 to RAM2 and likewise FLP_USE MDV will divert calls to any microdrive to the corresponding disk-drive.
It is not possible to divert calls to individual devices, ie divert calls for MDV2 to RAM3.
It is also possible to divert calls for ram-disks to the disk drives, and vice-versa,

```
eg          FLP_USE RAM.
```

N.B. When this command has been used, it is not possible to use the microdrives. To allow the microdrives to be used again, and the ram-disks or disk-drives to be called by their device names, the followuing type of command should be used:-

```
eg          RAM_USE RAM    (allows the microdrives to be used)
```

Using the rest of the Toolkit. (V 1.14 only)

If you are likely to want to use any of the following toolkit commands, they must be switched on after every reset or power up, using the following sequence of commands:-

```
FLP_EXT
NEW
```

LISTING EXTENSIONS

syntax EXTRAS (#channel)

As the Superbasic interpreter is extendable, the proceedure EXTRAS can be used to list any extra proceedures and functions linked into the interpreter. When the window is full, EXTRAS will freeze the screen (CTRL 5). EXTRAS will not list any proceedures which have been written in Basic by the user.

VIEW - EXAMINING A FILE

syntax VIEW (#channel,)(device_file name_extension)

VIEW is a procedure which allows a file to be examined in any window on the QL display. A channel to a window can be specified in the normal manner, or the channel can be omitted, and the file will be displayed on the default channel. The device name, from which the file is to be read, and the full file name, must be given. Lines may be truncated to fit in the window, and when the window is full, CTRL F5 is generated. If the file being VIEWed is to be edited, it will then have to be loaded into memory as normal.

eg VIEW MDV1_TEST_BAS will show the file TEST_BAS in the default window.

CLOCK - RESIDENT CLOCK

syntax CLOCK (#channel) default clock, 2 rows of 10 characters
CLOCK (#channel)string user defined clock

CLOCK is a procedure to set up the format of a resident digital clock. If no window is specified, a default window is set up in the top right corner of the monitor mode default channel 0. This window is 60 by 20 pixels, and is only suitable for four colour mode. The clock may be invoked to execute within a window set up by basic. In this case, the clock job will be removed when the window is closed.
The string is used to define the characters written to the clock window, and any character may be written except $ or %. If a dollar sign is found in the string, the next character is checked and %y or %Y will insert the two digit year
%d or %D will insert the two digit day of the month
%h or %H will insert the two digit hour
%m or %M will insert the two digit minute
%s or %S will insert the two digit second

The default string is "$d %d $m %h:%m:%s". A newline should be forced by padding out a line with spaces until the right hand margin of the window is reached. The correct time must be set using the SDATE command, as described in the QL manual.

eg CLOCK #2,"it is now %h:%m:%s on $d %d of $m, 19%y"
(pay attention to the spaces when setting the format!)

RENAME and TRUNCATE

The RENAME and TRUNCATE procedures operate on microdrive files if the Sinclair QL Toolkit has been added. They will operate, without the Sinclair QL Toolkit, on disk files and ram-disk files, but if used on microdrive files, the error message, "bad parameter" will be returned. The file name must include the device name where the file is held.

syntax RENAME (old file name),(new file name)

eg RENAME flp1_diary_doc,flp1_story_doc

syntax TRUNCATE (#channel) truncates the file which is open on the channel to the current file position.

WILD CARD COMMANDS

Wild card commands make use of variations in the filenames to be found in the directories on the storage mediums, which will be microdrives, ram-disks or floppy disks. A full filename will consist of two parts, the filename itself, and an extension, such as Test_bas, or boot_quill. If either part of the filename is left out, in a specified command, the remaining section will determine which files the command will operate on. Examples of the variations are given with the following commands.

WCOPY

syntax WCOPY device TO device
WCOPY device,device

The function of WCOPY is essentialy the same as COPY, except that where copy will only work on single files, WCOPY can be made to work on large numbers of files using just one command.
The simplest use is to copy, for example, complete microdrive cartridges to disk's. To do this, the above syntax applies, eg,

```
WCOPY mdv2_ to flp2_
```

The computer will then display the source filename and destination filename in the default window, and request the user to choose one of the following options:-

Y	(yes)	copy this file
N	(no)	do not copy this file
A	(all)	copy this, and all subsequent files
Q	(quit)	do not copy this, or any further files

If a destination file already exists, the user will be given the same options, but with the following meanings:-

Y	(yes)	copy this file, overwriting the old file
N	(no)	do not copy this file across
A	(all)	overwrite this file, and any further files which may require to be overwriten.
Q	(quit)	do not copy this, or any further files

WCOPY can also be used to copy groups of files, by using various parts of the filenames in the command. For example, to copy all the document files from one disk and no other files, to a back-up disk, you would use the following:-

```
WCOPY flp1__doc,flp2__doc
```

Note the double underscore, which means that the filename can be anything, but that the filename extension is required. To copy a batch of files with the same filename, and varying extensions, the following command would be used:-

```
WCOPY flp1_test_,flp2_test_
```

The underscore after boot, implies that the filename is required, but that the filename extension can be anything.

WDEL

syntax WDEL (#channel,)(wild_name)

The operation of WDEL is very similar to that of WCOPY in that each filename is displayed in the default channel, and you are given the following options:-

Y	(yes)	delete this file
N	(no)	do not delete this file
A	(all)	delete this, and all subsequent files
Q	(quit)	do not delete this, or any further files

An example of the use of WDEL would be to delete only DOC files on the disk in drive 2.

eg WDEL flp2__doc (note the double underscore which specifies the file extension)

There is a variation on the WDEL command which will delete all the files on the specified medium, or a batch of files, without asking for confirmation. An example of this is as follows:-

```
eg          WDEL_F flp2__doc     (forced deletion of doc files with
                                 no confirmation)
```

WDIR

```
syntax      WDIR (#channel,)(wild_name)
```

WDIR will list only the files from a specified medium , with a specific filename or extension. If the default window is filled, CTRL 5 will be generated.

```
eg          WDIR flp1__bas       (gives a partial directory of flp1
                                 containing files with a _bas
                                 extension)
```

WSTAT

```
syntax      WSTAT (#channel,)(wild_name)
```

WSTAT will list the file name, the file length, and when the file was last updated, of all the files on the specified device with the specified file name or extension. When the default window is full, CTRL 5 will be generated.

```
eg          WSTAT flp2_test_     (gives the above information about
                                 the files on flp2_ with a file
                                 name of test, regardless of the
                                 extensions)
```

A variation on WSTAT is STAT (#channel,)(name/device), which will give the name of the medium, the number of free sectors, and the total number of sectors on that medium. (Very similar to the report given after formatting a storage medium.)

A further variation on the wild card commands is that the storage device may be ommitted, as long as a default storage medium has been defined. This is done by means of a variation of the _USE command, ie DATA_USE flp2_ which will define flp2_ as the default data store. In this case, WSTAT test_ would now display the relevant information concerning all test files on flp2 regardless of their extensions, without having to specify flp2. After using DATA_USE flp2_ , other devices like flp1 would have to be specified in the wild card commands.

SPL, SPL_USE and DATA_USE

```
syntax      SPL source_file (TO destination_file)
            SPL_USE device_name:directory_name
(A device_name does not end in _: a directory_name must end in _)
```

The SPL command sets up a job to copy a file in the background, while the computer continues with its main task. The source and destination files may be given as either names or channels, or they can be ommitted and the command will use the default settings. These can be set with the USE command, ie, DATA_USE to set the source of the file to be spooled (default source is MDV2), and SPL_USE to set the destination of the file (default destination is set as SER1). When copying a file, SPL will not copy the file header if it is a data file, but it will copy the header for special files, ie executable program files. Also, if a destination file already exists, ie, an old version, it will be overwritten.

If the SPL command is given with only one parameter (the source filename), the output file (or device) will be derived from the default set by SPL_USE , as either

```
           1)   directory_name & source_name
or         2)   device_name
```

If the SPL command is given with two parameters, the output file (or device) will be derived as

```
           1)   destination_filename
or         2)   directory_name & destination_filename
```

The commonest use of SPL will be to copy files in the background, but it can be used as a true print spooler when used with the default device, SER1. If the output device is in use already, the SPL job will suspend itself until the device is available.

```
eg,        DATA_USE FLP2
           SPL testdata   this will spool the file, testdata, from
                          FLP2 to SER1.

           SPL ram2_test_bas,mdv1_test_bas    this will spool the
                                              file, test_bas, from
                                              ram2 to mdv1.

           SPL text to #3      this will spool the file, text, to
                               the device or file which has been
                               opened on channel 3.
```

MACHINE CODE EXTENSIONS

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

This variant of the OPEN call opens a file for read/write 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, ie, ram1_new_name.

TRUNCATE	This call truncates a file to the current byte position

JOBS, AJOB, SPJOB, RJOB - job control

As QDOS is a multitasking operating system, it is possible to have, at any one time, a number of competing or co-operating jobs, in the QL. These jobs will compete for resources, in line with their priority, and they may co-operate using pipes or shared memory to communicate. The basic attributes of a job are its position within the tree of jobs (ownership). A job is identified by two numbers, a job number which is an index in the table of jobs, and a tag, which is used to identify a particular job, so that it does not get confused with a previous job occupying the same position in the job table. In QDOS, these two numbers are combined into the job ID which is the job number + tag*65536. For the job control routines where the job ID is a parameter of the routine, it may be given as either a single number (Job ID, as returned from OJOB or NXJOB of the QL Toolkit) or as a pair of numbers (job number, job tag). For example, the parameter 65538, is equivalent to the parameters 2,1.

JOBS is a command which lists all the jobs running in the QL at that particular time, and gives the following information about each job:-

the job number
the job tag
the job's owner job number
a flag 'S' if the job is currently suspended
the job priority
the job (or program) name

syntax	JOBS (#n)	where n is the channel on which the information is to be displayed.

There are three procedures which allow you to control the jobs running in the QL, and these are,

AJOB job_id,priority	activates a job
SPJOB job_id,priority	sets a job's priority
RJOB job_id,error_code	removes a job from the QL

The procedure may fail if a job, or jobs, are removed from the QL while the procedure is listing them. If a job is waiting for another job to finish, and the second job has been removed from the QL, with RJOB, the first job will be released with D0 set to the error_code. If there are too many jobs ruuning in the QL for them all to be listed, the procedure will freeze the screen (CTRL S) when it is full.

FILE OPEN FUNCTION - FOPEN

This command, and four variations, provide a set of functions for opening files for different purposes. They differ from the OPEN procedure in the QL rom in that if a file error occurs, ie, "not found" or "already exists", the function will return the error code, and continue. Also, these functions use the DATA_USE directory default.

```
syntax    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  an
                                 old file
          FOP_DIR (#3,name)     open a directory
```

Directory entries may be read using GET to get the information, and the format of each entry is as follows:-

the length of each entry is 64 bytes, with the length of the file being given at the start of the entry, and the file name starting at the 14th byte, as a standard string. The update date of the file is given as a long integer, starting at the 56th byte.

Modifications to the disk drive parameters.

FLP_OPT

```
syntax    FLP_OPT security level(,start up time(,number of tracks))
```

The use of this command will allow the user to make use of older designs of disk drives and also set the security level for the disk system. Adjusting the security level of the system will also alter the access time.

Security level.

In the microdrive filing system each sector includes information which identifies the cartridge. This has the function of allowing the QL to check whether a cartridge has been changed, with each access. Most floppy disk formats do not allow this type of security and to comply with the filing system, identifying information is placed in sector 1 of track 0. The security level chosen will alter the extent to which the system may be abused by changing disks, while they are in use, without destroying the data on the disks.

Level 0. The disk will only be 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 will only be updated after a file is closed (or flushed) when half a second has elapsed without any other disk operation.

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

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, disks should only be changed when the motor is stopped. If files are open while a disk is changed, the read operations will be confused and the write operations will be aborted. This should maintain the integrity of the data on the disk.

Level 2. (Default Level)
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 been 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, orafter a FS.FLUSH call.

This is the default level and data should be secure unless a disk is changed while the motor is running.

Security system errors.

The two error messages which may be displayed on the screen are

```
'disk name'   read/write failed
```

which means that an attempt to read or write a sector on the disk has failed and

```
'disk name'   files still open
```

which means that a disk has been changed while it is still in use. If the system attempts to write to disk which has been changed, both error messages may be displayed indicating that attempt to write has been aborted and that files were still open when the disk was changed.

Start up time.

The system will always try to read data as soon as possible from the disk but to prevent errors and loss of data, there is always a delay in the read and write operations to allow the motor speed to stabilise. The system default for this delay is set to 0.6 seconds which is suitable for most modern drives and may be adjusted if required by giving an integer value for the start up time parameter.

Each increment in the parameter is equivalent to a delay of 20 milliseconds, giving a default value of 30. A value of 13 (260 milliseconds) may be found suitable for modern drives which have direct drive motors, while older drives may require a value of 60 (1.2 seconds). Values greater than 90 may cause problems with some systems as the motors will be stopping before the start up time has elapsed.

Number of tracks.

The QL disk format allows the number of tracks on a disk to be read from the disk itself and this number may be set when the disk is to be formatted. Initialy the system will check to see if there are at least 55 tracks on the disk. If there are, the system will assume 80 tracks, otherwise it will assume 40 tracks. The option to specify the number of tracks will allow the use of drives with 37 or 75 tracks and also save possible wear or damage to a 40 track drive when seeking track 55.

It is also possible to format a disk as single sided, on a double sided drive by making the eleventh character of the disk name an asterisk (*).

```
eg,        FORMAT 'FLP1_test files*'
```

This will give 720 sectors instead of 1440 sectors and the "*" will not appear in the disk name (invisible character).

Direct access to files.

GET,BGET,PUT,BPUT,FPOS

In QDOS, files appear as a continious stream of bytes and on a directory device (microdrives, floppy disks, etc...) the file pointer may be set to any position within a file, thus providing direct access to any data in that file. Such access may be read or write or both, depending on how the file was opened. Procedures have been provided to access single bytes, integers, floating point numbers, strings and also to locate the current file position.

```
syntax    'COMMAND' #channel \\ position,items
```

It is usual but not essential to specify a channel for the direct I/O commands (the default channel is #3). If no pointer is given, the routines will read or write from the current file position, otherwise the file position will be set before the list of items is processed. If the pointer is a floating point variable instead of an expression, it will be updated to the new position after all the items have been processed. If no items are given, nothing will be written to or read from the file. This can be used to position the file for use by other commands such as INPUT for formatted input.

Byte I/O.

```
syntax     BGET #channel \\ position,items          (get bytes from a file)
           BPUT #channel \\ position,items          (put bytes onto a file)
```

For BGET, each item must be a floating point or integer variable and for each item, a single byte will be fetched. For BPUT, each item must evaluate to an integer between 0 and 255 and a single byte will be sent for each item.

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

will put bytes 4, 12 and 243 after the current file position on the file which is open on channel #3.

If 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. Eg, if an EPSON compatible printer is set on channel 3, the printer could be put into condensed underline mode by

```
           BPUT #3,15,27,45,1
       or  BPUT #3,chr$(15);chr$(27);"-";chr$(1);
```

Unformatted I/O.

```
syntax     GET #channel \\ position,items      (get internal format data
                                                from a file)
           PUT #channel \\ position,items      (put internal format data
                                                onto a file)
```

The GET and PUT commands can be used for reading and writing data to and from files in the QL's internal format,ie 2 bytes for an integer(most significant byte first), 6 bytes for a floating point number (2 byte exponent to base 2, offset by 81H, followed by a 4 byte mantissa, normalised so that bits 31 and 30 are different) and a minimum of 2 bytes for a string variable (2 byte positive integer giving the number of characters in the string, followed by the characters).

For GET, each item should be an integer, floating point or a string variable and should match the data read from the file. For PUT, the data type should match the type of item in the parameter list.

```
Eg,        fpoint = 54
           john% = 42:salary = 78000:name$ = "Smith"

           PUT #3 \\ fpoint,john%,salary,name$
```

will position the file, open on channel 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 would then be set to 69.

For variables and array elements the type is self evident, while for expressions the type can be forced.

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

```
Eg,        xyz$ = "ab258.z"
           PUT #3 \\ 37,xyz$(3 to 5) || 0
```

will position the file opened on channel 3 to the 37th byte and then put the integer 258 on the file in the form of 2 bytes.

File position.

```
syntax     FPOS (#channel)     finds the current file position on the
                               specified channel.
```

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

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

The file pointer may be set by BGET, BPUT, GET or PUT with no items to be got or put. If an attempt is made to set the pointer beyond the end of the file, no error will be returned and the file pointer will be set to the end of the file.

N.B. setting the file pointer does not mean that the required part of the file is in a buffer but that the required part is being fetched. This means that an application could prefetch parts of a file, if the device driver is capable of prefetching.

File Enquiry Functions.

```
syntax     FLEN (#n)       returns the file length
           FTYPE (#n)      returns the file type (0=normal, 1=EXEC)
           FDAT (#n)       returns the data space for EXEC files
```

```
eg         OPEN #3,mdv1_fred:PRINT FLEN(#3)
```

will print the length of the file 'fred' on mdv1.

The three functions above allow the user to obtain information from the file header. In current versions of the microdrive handler, the header is only updated on an FS.HEADS call or when the file is closed. The floppy disk driver will also update 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 when the file was opened.

DIRECT SECTOR ACCESS

The facilities are also provided in the software to allow sectors to be writen and read directly, on a disk, without having to load a file into memory first. In order to do this, a special file must be opened on the disk with the name *Dnd

eg	FLP1_*Dsd	s is the sector length (0=128 bytes, 1=256 bytes, 2=512 bytes, 3=1024 bytes)
		d is the density (S=single density (FM) and D=double density (MFM))

The file name should be enclosed in quotes or apostrophes if it has been opened from Superbasic and no other file may be open on the drive if a direct read/write file is open.

Data may be read to or from the file using the GET or PUT commands as in the following example which will read sector 1 of track 2 on side 1.

eg	OPEN #3,'flp1_*D2d'	open file of specified type
	GET #3\1+0*256+2*65536,a$	read sector into a$

In order to specify the desired sector, the specification is as follows,

sector number + side * 256 + track * 65536

If the disk being read has been formated as a 40 track disk, the track number should be doubled.
If a read/write error is returned from a direct sector read/write call it will be safest to make a further call to track 0 as this will cause a restore and reset the drive to a known state.

An example program to read individual sectors is given below.

```
10 CLS:CLS#2
20 OPEN #3,'flp1_*D2d'
30 INPUT ("side = ");side
40 INPUT ("track = ");track
50 INPUT ("sector = ");sector
60 CLS:CLS#2
70 PRINT #2,"side= ";side;" ";"track= ";track;" ";"sector= ";sector
80 GET #3\sector+side*256+track*65536,a$
90 PRINT #2,a$
100 GO TO 30
```

N.B. disk sides are either 0 or 1
tracks are numbered from 0 to 79
sectors are numbered from 1 to 9

OPERATING NOTES

Using PSION - QUILL,EASEL,ABACUS, and ARCHIVE

To use the Psion packages with the RamDisk software, the following changes must be made to all four packages.
(Versions 2.0 and upwards, are the only versions which can run on the expanded QL)

Insert Psion package in microdrive 1 (MDV1)

```
          LOAD MDV1_BOOT
          EDIT 9
```

Change the SCR to CON

```
          DELETE MDV1_BOOT   (This deletes the original boot file)
          SAVE MDV1_BOOT     (This saves the modified boot file)
```

This procedure should be carried out on all four Psion packages.

NOTE: This will not affect the operation of any of the Psion packages when using them with microdrives.

If it is intended to use the Psion packages from the disk drives, it can be done with no further modifications than those described above. See the instructions under USE.

If it is required to have access to the microdrives at the same time as the disk drives, the following modification should be done at the same time as the RamDisk software modification.

After editing line 9,

```
          EDIT 8
```

Change MDV to FLP
Save the corrected BOOT file as described above.

The package may now be run using LRUN FLP1_BOOT, or it will automaticaly boot from FLP1 after the QL has been reset.

It may be required that all four Psion packages be on the same disk. In this case, the BOOT files must be saved with an extension, eg, BOOT_QUILL, followed by copying the remaining Psion files to the floppy disk. In this case, the required package would have to be run using LRUN FLP1_BOOT_QUILL or for Abacus, LRUN FLP1_BOOT_ABACUS.

Saving and loading files is still the same as with an unexpanded QL, except that you can now specify the destination, or source, of the files as microdrives, floppy disk's, or Ram-Disk's.