

SUPERQBOARD

USER MANUAL

HP

Every effort has been made to ensure that the information given in this manual is as accurate as possible, however, Sandy S.r.l. will not accept any responsibility for losses or damage caused directly or indirectly from its use.

Copyright Sandy S.r.l. - Milan
Copyright Tony Tebby Q-Jump

QL and QDOS are trademarks of Sinclair Research Limited

Introduction

- INTRODUCTION
- INSTALLATION
- FLOPPY DISK DRIVER
- RAM DISK DRIVER
- PRINTER PORT
- BASIC EXTENTIONS

Introduction

This document describes the facilities included with the SANDY disk interface. Throughout this document the following conventions are used:

COMMAND *for parts of commands to be typed as shown*

example *for parts of commands shown as examples*

description *for descriptive parts of commands.*

The SANDY interface has the hardware and software to add a floppy disk system, a parallel printer port, and 512k bytes of RAM (including a RAM disk) to the QL. In addition it has all the file handling facilities of the Sinclair QL Toolkit together with some other extensions to SuperBASIC.

Contents

Floppy Disk Driver	1
RAM Disk Driver	10
Parallel Printer Port	15
SuperBASIC Extensions	
SPL SPL_USE - File Spooler	19
JOBS AJOB SPJOB RJOB - Job Control	21
GET BGET PUT BPUT FPOS - Direct Access Files	22
FLEN FTYP FDAT - File Enquiry Functions	25
FOPEN FOP_IN FOP_NEW FOP_OVER FOP_DIR	26
VIEW - Examining a file	27
WDIR WSTAT WDEL WDEL_F - Wild Card Commands	28
STAT - Drive Statistics	29
WCOPY - Wild Card Copy	30
RENAME TRUNCATE	32
DATA USE - Default Directory	33
CLOCK - Resident Clock	34
EXTRAS - Listing Extensions	35

Installation

INTRODUCTION

INSTALLATION

FLOPPY DISK DRIVER

RAM DISK DRIVER

PRINTER PORT

BASIC EXTENTIONS

AFTER FITTING YOUR SUPERQBOARD AND SWITCHING ON THE POWER THE START-UP MESSAGE DISPLAYS THE TOTAL MEMORY SIZE. SOME QL VERSIONS WILL RETURN A 128K MEMORY SIZE AT POWER UP - JUST PRESS RESET TO LINK THE FULL 640K.

TO LINK IN THE SUPERTOOLKIT II EXTENSIONS TO BASIC USE THE COMMAND "TK2_EXT" THIS CAN BE ENTERED DIRECTLY OR ADDED TO YOUR BOOT PROGRAMS.

DIAGRAM 1 - SUPERBOARD INTERFACE

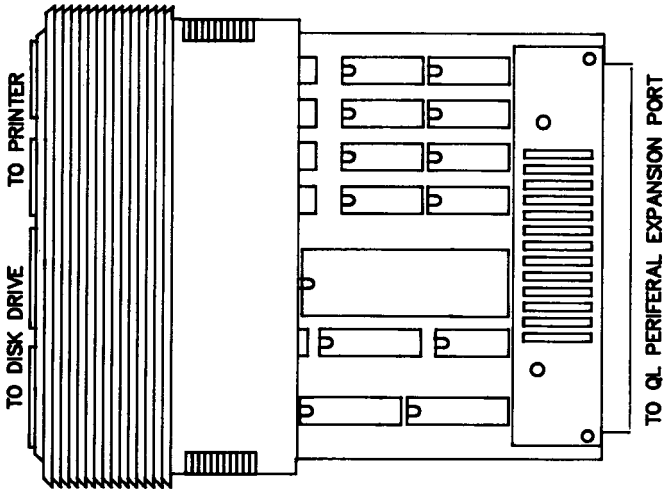


DIAGRAM 2 - PIN CONNECTIONS

PRINTER



PIN	CODE
01	STROBE
02	DATA 1
03	DATA 2
04	DATA 3
05	DATA 4
06	DATA 5
07	DATA 6
08	DATA 7
09	DATA 8
10	ACKNLG
11	BUSY
12	N.C.
13	N.C.
14/24	GND
28	N.C.

DISK DRIVE



PIN	CODE
02	N.C.
04	N.C.
06	N.C.
08	INDEX
10	DRIVE A
12	DRIVE B
14	N.C.
16	MOTOR ON
18	DIRECTION
20	STEP
22	WRITE DATA
24	WRITE GATE
26	TRACK 00
28	WRITE PROTECT
30	READ DATA
32	SIDE 1
34	N.C.
01/33	GND

FITTING INSTRUCTIONS

1. Fitting the interface card.

DISCONNECT THE QL AND ANY OTHER PERIPHERAL FROM THE MAINS.

Locate the peripheral expansion port on the QL (on the far left) and gently remove the plastic cover by pulling it away from the QL. Be patient since this may take some effort.

Now place the QL upright with the expansion port upwards, grasp the Super Q Board by its plastic cover and gently slide it downwards into position - the components of the Super Q Board should be on the same side as the QL keyboard.

The interface card is firmly in position when the plastic cover is flush against the side of the QL.

2. Connecting the disk drives

DISCONNECT THE QL AND ANY OTHER PERIPHERAL FROM THE MAINS.

You will notice that the Super Q Board has two ports, a small one and a large one. The large port is where you should fit your disk drive connector (the one on the end of a ribbon cable). The connector will only plug in one way up, so do not force it.

3. Connecting the printer

As stated above the Super Q Board has two ports, the small one is the centronics printer port. A suitable cable can be supplied by your dealer. Once again the connector plugs in one way only.

4. Testing the Super Q Board

Power up sequence:

1. Switch on QL
2. Switch on Disk drives

At this point, the interface will check if there is a floppy disk in the disk drive (no.1 if you have a twin unit) and then stop the drive motor, and the message :

SANDY SUPER Q BOARD V.1.16 (or later) copyright 1984

with the Sinclair TV/Monitor (F1-F2) prompt will appear.

At this point you may press F1 or F2 accordingly.

The interface will check if there is a file called "BOOT" on the disk, if no file called "BOOT" is found the drive motor will stop.

If you did not put any floppy disk in before power up, the interface will still look for a floppy on drive1, stop the motor and then after you have pressed F1 or F2 will turn the microdrive motor.

Power down sequence:

1. Switch off Disk drives
2. Switch off QL

5. Removing the interface card

If for any reason you need to remove the interface card, remember to:

DISCONNECT THE QL AND ANY OTHER PERIPHERAL FROM THE MAINS.

Unplug the disk and printer cables, grasp the Super Q Board by the plastic cover and pull firmly.

Floppy Disk Driver

CONTENTS

Beginners Start Here	1
Floppy Disk Compatibility	2
Auto-Boot	2
Microdrive Emulation	3
Floppy Disk Options	3
Security	4
Security Level 0	5
Security Level 1	5
Security Level 2	5
Security System Errors	6
Start Up Time	6
Number Of Tracks	6
Direct Sector Read/Write	7
Disk Drive Specification	8

INTRODUCTION

INSTALLATION

FLOPPY DISK DRIVER

RAM DISK DRIVER

PRINTER PORT

BASIC EXTENTIONS

QJUMP Floppy Disk Driver 1.16

Beginners Start Here

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:

```
100 FLP_USE mdv
110 CLOSE #1: CLOSE #2
120 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_200	<i>formats a new floppy disk in drive 1</i>
DIR flp1_	<i>directory listing of floppy disk 1</i>
SAVE flp1_myprog	<i>save the current SuperBASIC program as 'myprog' in floppy disk 1</i>
OPEN_NEW #3,flp2_data	<i>creates and opens a new file 'data' in floppy disk 2</i>
COPY mdv1_x TO flp1_x	<i>copies file x from Microdrive 1 to floppy disk 1</i>

Floppy Disk Compatibility

The QJUMP Floppy Disk driver software provides easy upgrade path from Microdrives to floppy disk speed and storage capacity. It not only provides all the built-in Microdrive filing system operations, but includes all the extended filing system operations provided in the Sinclair QL Toolkit for Microdrives. This allows all the SuperBASIC extensions provided in the QL Toolkit (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. FLPI_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.

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 driver also includes a SuperBASIC procedure FLP_USE to change the name of the floppy disk driver.

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

resets the name of the floppy disk driver to 'mdv', so that all subsequent open calls for Microdrives will use the floppy disks instead. Thus the commands

```
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

```
FLP_USE flp
```

will reset the driver to its normal state.

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.

these parameters are specified by three separate commands each with one parameter:

FLP_SEC security level
FLP_START start up time
FLP_TRACK nr 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. On some floppy disk systems, if an attempt is made to access a drive which has no disk in place, or has the door open, then the motors will continue to run. If a drive select light is still on, then the motors may be stopped by inserting a disk and closing the door. In any case, the drives are deemed to be stopped if 5 seconds have passed without a disk access.

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). A value of 90 (1.8 seconds) or more may cause problems with some disk systems, as the motors may stop automatically before the start up time has elapsed!

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

FLP1_*Dsd where *s* is the sector length
 0=128 bytes, 1=256 bytes, 2=512 bytes, 3=1024 bytes

 and *d* is the density
 S=single (FM), 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.

If the attempt to read or write a sector fails, D0 will be returned as a standard error message pointer (read / write failed).

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) or
 ignored (IO.SSTRG)

2 + sector len returns the sector length followed by the
 sector (IO.FSTRG) or
 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 QL Toolkit, 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	40T+40T	80T	80T+80T
Drive				
40T	C	?	X	X
40T+40T	C	C	X	X
80T	R	?	C	?
80T+80T	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 'f1p1_disk_name *'
```

Ram Disk Driver

CONTENTS

Beginners Start Here	10
Ram Disk Compatibility	11
Ram Disk Creation	11
Heap Fragmentation	12
Microdrive Emulation	12
Examples	13

INTRODUCTION

INSTALLATION

FLOPPY DISK DRIVER

RAM DISK DRIVER

PRINTER PORT

BASIC EXTENTIONS

QJUMP 'RAM Disk' Driver 1.01

Beginners Start Here

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 term 'RAM disk' is a misnomer. It is used to denote a 'virtual' device (one that one only exists in the fertile imagination of the QL) that looks and behaves like a very fast disk device. It is so fast because being virtual, there is virtually nothing to move to get information in and out. It is, in fact, no more than a reserved area of the QL's main memory (its RAM - Random Access Memory). This means, of course, that any space taken by a RAM disk is not available to programs executing in the QL. Furthermore, any data stored in a RAM disk will be lost when the QL is turned off or reset!

RAM disks in the QL may be of any size, subject to there being enough memory. The normal usage of a RAM disk would be to copy all working files from Microdrive (or floppy disk) into a RAM disk; rename the RAM device to be MDV (to pretend that the data is really on the Microdrives); execute the programs (e.g. Quill, Archive etc.); and, at the end of the session, rename the RAM device to be RAM before copying the data files back to Microdrive.

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

FORMAT ram2_200	<i>creates a new RAM disk 2, see below</i>
DIR ram1_	<i>directory listing of RAM disk 1</i>
SAVE ram1_myprog	<i>save the current SuperBASIC program as 'myprog' in RAM disk 1</i>

OPEN_NEW #3,ram2_data *creates and opens a new file 'data'
 in RAM disk 2*

COPY mdv1_x TO ram1_x *copies file x from Microdrive 1 to
 RAM disk 1*

RAM Disk Compatibility

The QJUMP RAM Disk driver software provides a means of instant access data storage using the standard IO system calls. It not only provides all the built-in Microdrive filing system operations, but includes all of the Microdrive extensions provided in the Sinclair QL Toolkit. This allows the use of the extended filing system commands which are provided in the QL Toolkit, and these may also be accessed from assembly language code with the following calls.

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. RAM1_NEW_NAME).

TRUNCATE Trap 3, D0=4B

This call truncates a file to the current byte position.

RAM Disk Creation

There are two forms of 'RAM disk' for the QL: in one form the space for the files in the RAM disk is allocated dynamically using any spare memory in the QL. Unfortunately, this scheme, although very simple to implement using QDOS, does not work in conjunction with Psion programs or any other programs which automatically use all the spare memory themselves. However, this other form of RAM disk has its own memory allocation routines which operate within a predefined area of the QL's memory.

A 'RAM disk' is created by formatting it: the size, in sectors, is given in place of the usual medium name.

```
FORMAT ram2_80
```

removes the old RAM disk number 2, and sets up a new RAM disk of 80 sectors. A RAM disk may be removed by giving either a null name or zero sectors

```
FORMAT ram1_ or FORMAT ram1_0
```

The RAM disk number should be between 1 and eight, inclusive, while the number of sectors (512 bytes) is limited by the memory available.

Heap Fragmentation

The primary storage mechanism in the QL for permanent or semi-permanent memory allocations is a 'heap'. Allocating space in a heap, and then re-allocating this space as a different size, inevitably causes holes to be left within the heap. This reduces the amount of memory available to either SuperBASIC or executable programs.

This RAM disk driver has precautions to reduce the possibility of heap fragmentation, but it is preferable to consider any RAM disk to be a permanent feature until the QL is reset. Supertoolkit II from QJUMP has a command DEL_DEFB which will perform a limited 'garbage collection' on a fragmented heap.

Microdrive Emulation

The standard driver also includes a SuperBASIC procedure RAM_USE to change the name of the RAM disk driver.

```
RAM_USE mdv or RAM_USE 'mdv'
```

resets the name of the RAM disk driver to 'mdv', so that all subsequent open calls for Microdrives will use the RAM disks instead. Thus the commands

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

will actually open the file 'myfile' on RAM 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

RAM_USE ram

will reset the driver to its normal state.

Examples

The following example will copy selected files from a microdrive cartridge to a RAM disk as well as copying QUILL (V2.3, other versions of QUILL may require a different size of RAM disk).

```
100 PRINT #0,'Put QUILL in MDV1 and press a key'
110 PAUSE
120 FORMAT ram1_150
130 COPY mdv1_quill,ram1_quill
140 COPY mdv1_quil_hob,ram1_quil_hob
150 COPY mdv1_printer_dat,ram1_printer_dat
160:
170 PRINT #0,'Put data cartridge in MDV2 and press a key'
180 PAUSE
190 FORMAT ram2_200
200 OPEN_NEW #3,ram2_file_list: REMark make list of files
210 DIR #3,mdv2_
220 CLOSE #3
230 OPEN IN #3,ram2_file_list
240 INPUT #3,a$,a$: REMark skip heading
250 REPEAT files
260 IF EOF(#3): EXIT files
270 INPUT #3,file$
280 INPUT #0,'Copy '&file$&' to RAM disk? ';ans$
290 IF 'y' INSTR ans$: COPY 'mdv2_ '&file$ TO 'ram2_ '&file$
300 ENDREP files
310 CLOSE #3
320 DELETE ram2_file_list
330 :
340 RAM_USE mdv: REMark all across from mdv
```


This program copies files back to microdrive at the end of a session.

```
100 RAM_USE ram:          REMark reset RAM name
110 :
120 PRINT #0,'Put data cartridge in MDV2 and press a key'
130 PAUSE
140 OPEN_NEW #3,ram2_file_list: REMark make list of files
150 DIR #3,ram2_
160 CLOSE #3
170 OPEN_IN #3,ram2_file_list
180 INPUT #3,a$,a$:      REMark skip heading
190 REPEAT files
200 IF EOF(#3): EXIT files
210 INPUT #3,file$.
220 IF file$='file_list': NEXT files
230 DELETE 'mdv2_'&file$
240 COPY 'ram2_'&file$ TO 'mdv2_'&file$
250 ENDREP files
260 CLOSE #3
270 DELETE ram2_file_list
```

Printer Port

CONTENTS

Beginners Start Here	15
More Power To Printer	16
Ser Emulation	17
Multiple Buffering	18

INTRODUCTION

INSTALLATION

FLOPPY DISK DRIVER

RAM DISK DRIVER

PRINTER PORT

BASIC EXTENTIONS

Parallel Printer Port V1.02

Beginners Start Here

All devices which connect your QL to the world outside its black case are identified to QDOS by a name. The serial ports (marked SER1 and SER2 on the case) are identified by the name SER, the network port (marked NET on the case) is identified by the name NET. Likewise the Parallel Printer Port is identified by the name PAR.

Most programs which send output to a printer are written to use the serial port SER1. The simplest way of changing over to using the parallel printer port is to include the command 'PAR USE SER' in any BOOT file, or to type this command just after the QL has been reset. This command will make the Parallel Printer Port pretend that its name is SER, fooling programs into sending their print output to the Parallel Printer Port instead of sending it to SER1.

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

```
100 PAR USE ser
110 CLOSE #1: CLOSE #2
120 EXEC_W mdv1_quill
....
....
```

The Psion programs, and many others, can be configured to change the name of the printer device. If this is to be done, it is only necessary to set the name of the printer to 'PAR' in the Psion and other programs which have internal configuration for various types of printers. Some other programs may require the name of the printer device to be set to 'PARC' for use with most daisywheel printers.

It is possible to make the Parallel Printer Port use a large buffer within the QL to make printing more efficient, but this technique will not work with the Psion programs which tend to grab most of the QL's memory for themselves, and it is most effective when used with additional memory.

More Power to Your Printer

The QJUMP driver for the parallel printer port provides several advantages over the SER drivers provided with the QL.

Large buffers may be specified to allow efficient print spooling.

A form feed may be created automatically when the channel is closed.

While only one channel may be open to the parallel port at one time, many complete print files may be held pending without tying up the port.

A channel is opened to the parallel printer port in exactly the same way as a channel is opened to one of the QL's serial ports. The only differences lie in the name that is used, and the options which are accepted as part of that name.

The specification of the device name is

PARcf_nk the 'c' flag is used if <CR> is required as the newline character

the 'f' flag is used if a form feed is required when the channel is closed

_n is the buffer size in bytes (up to 32767) unless

the 'k' flag is used to specify a buffer size in kilobytes.

For example

PAR parallel printer port with default buffer

PARF_3K ... with form feed at end of file and a 3 kilobyte buffer, resets the default buffer length to 3 kilobytes.

PARC_400 ... with <CR> in place of <LF> as newline and a 400 byte buffer, resets the default buffer length to 400 bytes.

The initial default buffer length is 128 bytes.

SER Emulation

There is an additional SuperBASIC command to make it possible to get the benefits of the Parallel Printer Port, without any need to change existing programs that use one of the QL's SER ports for printer output. This is the PAR_USE command.

PAR_USE SER

will make the Parallel Printer Port recognise the device name 'SER' in addition to the device name 'PAR'. The normal SER options of port number, parity (O, M, E or S), handshaking (I or H), and protocols (R or Z) are ignored, while the PAR options are recognised. In particular the default buffer length, set when a PAR channel is opened with a specified buffer length, is use for this pseudo-SER device.

For Example:

```
OPEN #3,PAR_10K: CLOSE #3: REM reset default buffer
PAR_USE SER
...
...
OPEN #3,SER1C
```

will open a channel to the Parallel Printer Port with a 10 kilobyte buffer and <CR> in place of <LF>, in the same way as the command:

```
OPEN #3,PARC_10K
```

This facility is disabled by re-specifying the PAR_USE to exclude the name SER. This will return the serial ports to their normal usage:

```
PAR_USE PAR
```

Multiple Buffering

To illustrate the multiple buffer capability of the printer driver, connect a printer and set it 'off line', then type in and run the following program:

```
100 FOR try=1 to 4
110 OPEN #3,parf_2k
120 FOR lno=1 to 50: PRINT #3,'Line ';lno;' of try ';try
130 END FOR try
140 CLOSE #3
```

When the printer is turned 'on line' the printer should print each 'try' in turn, showing the way in which the multiple outputs are queued, rather than confused, inside the Parallel Printer Port driver.

Basic Extensions

CONTENTS

File Spooler	19
Spl Examples	20
Job Control	21
Direct Access Files	22
File Open Functions	26
File Enquiry Functions	25
File Position	24
Examining a File	27
Wild Card Commands	28
List Drive Statistics	29
Wild Card Copying	30
Rename End Truncate	32
Data File Default	33
Resident Clock	34

INTRODUCTION

INSTALLATION

FLOPPY DISK DRIVER

RAM DISK DRIVER

PRINTER PORT

BASIC EXTENSIONS

SPL SPL USE - File Spooler

The SPL command sets up a job to copy a file. Only the source need be given: the destination may be defaulted. The source file has the default set up by the DATA_USE command. As supplied, the default destination is SER. The SuperBASIC interpreter will continue after the Job has been set up, with the file being copied in the background. SPL differs from COPY not only in that it operates as a job in the background, but also in the handling of file headers. The COPY procedure copies both the file and its header: to copy a file to a device like a printer, the variant COPY_N is used to copy without the header. SPL will, however, not copy the header from an ordinary data file, but it will copy the header of a file which is one of the special types (e.g. executable program file). Furthermore, when using SPL to copy from file to file, if the destination file already exists, then it will be overwritten.

The command syntax is

SPL *source_file* or

SPL *source_file* TO *destination*

The source and destination files may be given as names, or as a SuperBASIC channel number (e.g. 3).

The default set by the DATA_USE command is used to find the source file, and there is a special command, SPL_USE, to set the default destination. The default destination device or directory may be up to 32 characters long.

SPL_USE *device_name* or

SPL_USE *directory_name*

A *device_name* does not end in '_': a *directory_name* must end in '_'.

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

- 1) *directory_name* & *source_filename* or
- 2) *device_name*

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

- 1) destination_filename or
- 2) directory_name&destination_filename

SPL will often be used to copy files in the background, but it can be used as a true spooler when used with the default output device. In this case, if the output device is in use, the SPL job will suspend itself until the device is available.

SPL Examples

SPL myfile	<i>using the supplied defaults this will spool FLP2_MYFILE to SER</i>
SPL flp1_demo_myfile TO ser2	<i>the file FLP1_DEMO_MYFILE will be spooled to SER2</i>
DATA_USE flp2_demo SPL_USE ser2 SPL myfile	<i>this will also spool the file FLP2_DEMO_MYFILE to SER2</i>
SPL mdv2_myfile, mdv1_myfile	<i>does the obvious</i>
SPL_USE mdv1_ SPL myfile	<i>using the supplied DATA_USE default, this will also spool FLP2_MYFILE to MDV1_MYFILE</i>
SPL myfile TO #3	<i>will spool myfile to the file or device already opened as #3</i>

JOBS AJOB SPJOB RJOB - Job Control

As QDOS is a multitasking operating system, it is possible to have, at one time, in the QL a number of competing or co-operating jobs. Jobs 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 priority and its position within the tree of jobs (ownership). A job is identified by two numbers: one is the job number which is an index into the table of jobs, and the other is a tag which is used to identify a particular job so that it cannot be confused with a previous job occupying the same position in the job table. Within QDOS the two numbers are combined into the job ID which is job number + tag*65536. For these job control routines, where job_id is a parameter of one of the job control routines, it may be given as either a single number (the job ID, as returned from OJOB or NXJOB of the QL Toolkit) or as a pair of numbers (job number, job tag). Thus the single parameter 65538 (2+1*65536) is equivalent to the two parameters 2,1.

JOBS is a command to list all the jobs running in the QL at the time. If there are more jobs in the machine than can be listed in the output window, the procedure will freeze the screen (CTRL F5) when it is full. The procedure may fail if jobs are removed from the QL while the procedure is listing them. The following information is given for each job:

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

The syntax of the JOBS command is:

```
JOBS                                list jobs to window #1
JOBS #channel                       list jobs to a given channel
```

There are also three procedures for controlling Jobs in the QL:

```
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.
```

If there is a job waiting for the completion of a job removed by RJOB, it will be released with D0 set to error_code. E.g.

```
RJOB 2,1,0                          remove job 2 (tag 1) with no error
```

GET BGET PUT BPUT FPOS - Direct Access Files

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.

To keep files tidy there is a command to truncate a file (when information at the end of a file is no longer required), and a command to flush the file buffers.

A direct access input or output (I/O) command specifies the I/O channel, a pointer to the position in the file for the I/O operation to start and a list of items to be input or output.

command #channel\ position, items

It is usual (although not essential - the default is #3) to give a channel number for the direct I/O commands. If no pointer is given, the routines will read or write from the current position, otherwise 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 new current file position. If no items are given then nothing is written to or read from the file. This can be used to position a file for use by other commands (e.g. INPUT for formatted input).

Byte I/O

BGET #channel\ position, items *get bytes from a file*
BPUT #channel\ position, items *put bytes onto a file*

BGET gets 0 or more bytes from the channel. BPUT puts 0 or more bytes into the channel. For BGET, each item must be a 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 on the file open on channel #3.

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 be 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?

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\ position, items *get internal format data
from a file*

PUT #channel\ position, items *put internal format data
onto a file*

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.

```
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$='ab258.z'
...
PUT #3\37, xyz$(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. $1*256+2$).

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)      find file position
```

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 the commands BGET, BPUT, GET or PUT 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 prefetch of parts of a file where the device driver is capable of prefetching.

FLEN FTYP FDAT - File Enquiry Functions

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\ 99999)

FLEN (#n)	<i>returns the file length,</i>
FTYP (#n)	<i>returns the file type (0=normal 1=EXEC),</i>
FDAT (#n)	<i>returns the data space for EXEC files.</i>

```
OPEN #3, mdv1 fred PRINTs the length of file fred on mdv1_  
PRINT FLEN(#3)
```

FOPEN FOP_IN FOP_NEW FOP_OVER FOP_DIR - File Open Functions

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 (#channel, name)    open for read/write
FOP_IN (#channel, name)  open for read only
FOP_NEW (#channel, name) open a new file
FOP_OVER (#channel, name) open a new file, or overwrite old file
FOP_DIR (#channel, 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 14th byte of the entry giving the filename and there is the update date as a long integer starting at the 56th 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$&'_ASM') :REMark try to open _ASM file
IF ferr=-7: ferr=FOP_IN (#3,name$) :REMark ERR.NF, try no _ASM
```

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 (in #1): lines are truncated to fit in the window and, when the window is full, CTRL F5 is generated.

VIEW #*window*, *name* view a file in given window: the DATA_USE directory default is used.

WDIR WSTAT WDEL WDEL_F - 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).

The general forms of these commands are

command or
command #channel or
command wild name or
command #channel, wild name

The commands are

WDIR *list directory, generates CTRL F5 when the window is full.*

WSTAT *list file name, length and last update date, generates CTRL F5 when the window is full.*

WDEL *delete files (requests confirmation).*

WDEL_F *delete files (forced).*

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 wild_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 wild 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 wild 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 flp2_freda_list
_fred	flp2_fred flp2_freda_list flp2_old_fred flp2_old_freda_list
flp1_old_list	flp1_old_jo_list flp1_old_freda_list

Examples

WDEL	<i>delete all files in current directory, requesting confirmation</i>
WDIR #3,_asm	<i>list the names of all the assembler source files to channel #3</i>
WSTAT fred	<i>list the file statistics of all files beginning with fred</i>

STAT - List drive statistics

The STAT command has the same format as the wild card commands, but lists the statistics of the drive containing the file(s) referred to. These statistics are the medium name, the free sectors and the total good sectors.

STAT	<i>list the statistics of the current default</i>
STAT #2, mdv1_	<i>list the statistics of mdv1_ to channel #3</i>

WCOPY - Wild Card Copying

The WCOPY command has several optional forms:

WCOPY *source wild name* TO *destination wild name*
WCOPY *source wild name, destination wild name*
WCOPY #*channel*, *source wild name* TO *destination wild name*
WCOPY #*channel*, *source wild name, destination wild name*

If no channel is given, the dialogue will be in channel #1.

When using WCOPY, each source and destination filename is written to the chosen channel, and the user is requested to press one of:

Y (yes) copy this file
N (no) do not copy this file
Q (quit) do not copy this or any more files
A (all) copy this and all the next matching files.

If the destination file already exists, the user is requested to press one of:

Y (yes) copy this file, overwriting the old file
N (no) do not copy this file
Q (quit) do not copy this or any more files
A (all) overwrite the old file, and overwrite any other files requested to be copied.

WCOPY may be used to copy whole directories. The destination name is made up from the actual source file name and the destination wild name. If a missing section of the source wild name is matched by a missing section of the destination wild name, then that part of the actual source file name will be used as the corresponding part of the actual destination name. Otherwise the actual destination file name is taken from the destination wild name. If there are more sections in the destination wild name than in the source wild name, then these extra sections will be inserted after the drive name, and vice versa.

For example, if the default data directory is flp2_, then

WCOPY flp1_,flp2_ would copy all files on flp1_ to flp2

WCOPY fred,mog would copy

flp2_fred to flp2_mog
flp2_freda_list to flp2_moga_list

WCOPY _fred,_mog would copy

flp2_fred to flp2_mog
flp2_freda_list to flp2_moga_list
flp2_old_fred to flp2_old_mog
flp2_old_freda_list to flp2_old_moga_list

WCOPY _list,old_list would copy

flp2_jo_list to flp2_old_jo_list
flp2_freda_list to flp2_old_freda_list

WCOPY old_list,flp1_list would copy

flp2_old_jo_list to flp1_jo_list
flp2_old_freda_list to flp1_freda_list

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 name, new name* *renames a file, the DATA_USE default directory is used for both filenames.*

TRUNCATE #n *truncates the file open on #n to the current file position.*

RENAME fred, fred_back *renames file fred in the current directory as fred_back*

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)
```

will produce a directory listing of all filenames starting with `'flp1_project5_library'` and then 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'`)

CLOCK - Resident Clock

There are a number of optional forms of the CLOCK command

CLOCK	<i>default clock, 2 rows of 10 chars in default position</i>
CLOCK #channel	<i>default clock in defined channel</i>
CLOCK string	<i>user defined clock in default position</i>
CLOCK #channel, string	<i>user defined clock in defined channel</i>

CLOCK is a procedure to set up a resident digital clock. If no window is specified, then a default window is set up in the top RHS 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: any character may be written except \$ or %. If a dollar sign is found in the string then the next character is checked and

\$d or \$D will insert the three characters of the day of week,
\$m or \$M will insert the three characters of the month.

If a percentage sign is found then

%y or %Y will insert the two digit year
%d or %D will insert the two digit day of 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.

Example:

```
MODE 8
OPEN #6, 'scr_156x10a32x16'
INK #6,0: PAPER #6,4
CLOCK #6, 'QL time %h:%m'
```

EXTRAS - Listing All Extensions

EXTRAS or
EXTRAS #channel

The SuperBASIC interpreter is extendable. The procedure EXTRAS may be used to list the extra procedures and functions linked into the interpreter. EXTRAS will freeze the screen (CTRL F5) when the output window has been filled.

TOOLKIT Versions

New versions of Toolkit differ from Version 2.00 in the following respects:

BREAK (CTRL SPACE) is checked during **WCOPY** and **WREN** even if **A(ILL)** has been requested.

PRINT_USING and **FXP\$** have been added. **PRINT_USING** is more comprehensive than the form given in the draft manual.

The network file server has been extended to include serial device (printer) serving, as well as **QL-QL**

messaging. The **NFS_USE** command has been changed to give more flexibility, in particular several users may now share a data disk when using **QUILL**.

The **MG ROM** patch, which is not required for English language ROMs has been omitted, to make room for the above.

The network file serving protocol of Version 2.0 is not compatible with new Versions.

Obligatory Notice

QL, QDOS and **SINCLAIR** are trade marks of Sinclair Research Limited.

Copyright Tony Tobby 1985

All rights reserved. No part of this software or documentation may be reproduced in any form. Unauthorised copying, hiring, lending or sale and repurchase is prohibited.

Disclaimer:

In no circumstances will either Care Electronics or QJump be liable for any direct, indirect or consequential damage or loss including but not limited to loss of use, stored data, profit or contracts which may arise from any error, defect or failure of the ROM version of Super Toolkit II.

Care Electronics or Qjump has a policy of constant development and improvement of the products and will always inform the legitimate and registered user of this product about the changes and improvements the product is subject to.

User manual, English edition written by Tony Tobby, Qjump, UK.

QL, QL NET, QDOS and Superbasic are Trademarks of Sinclair Research Limited, UK.

INDEX OF CONTENTS

Preface		Section 15:	Memory Management
Section 1:	Introduction	Section 16:	Procedure Parameters
Section 2:	Contents of Toolkit II	Section 17:	Error Handling
Section 3:	File Editing	Section 18:	Time Keeping
Section 4:	Directory control	Section 19:	Extras
Section 5:	File Maintenance	Section 20:	Console Driver
Section 6:	SuperBasic programs	Section 21:	Microdrive Driver
Section 7:	Load and Save	Section 22:	Network Driver
Section 8:	Program Execution	Section 23:	Writing programs to use with EX
Section 9:	Job Control	Appendix A:	SuperBaic Extensions in alphabetic order and list of differences with previous versions of Toolkit
Section 10:	Open and Close	Appendix B:	Using the Toolkit II facilities with the GST assembler and linker
Section 11:	File Information	Appendix C:	QL Network protocols: Standard QL handshake and Toolkit II Broadcast
Section 12:	Direct Access Files		
Section 13:	Format Conversions		
Section 14:	Display Control		

PREFACE

The original QL Toolkit was produced in something of a rush to provide useful facilities which, arguably, should have been built in to the QL to start with. Since its appearance, I have been subject to continuous pressure to modify certain facilities and extend the range of facilities provided.

QLToolkit II is, therefore, a revised (to the extent of being almost completely rewritten) and much enlarged version of the original QLToolkit. Old facilities now work faster and are more compact, so that there is room in the ROM cartridge for over 100 operations.

The fact that QLToolkit II ever saw the light of day is due to prompting from a number of quarters. Many people have contacted me complaining that they have

been unable to lay their hands on the original QLToolkit, and this eventually convinced me that there was a market for a second version. Repeated criticism of the original facilities made at great length (and with justification) by Chas Dillon have provided the basis for many of the modifications to the old routines. Ed Bruley has provided invaluable practical support in putting the product on the market, and Cambridge Systems Technology allowed me to use one of their Winchester disc systems to test the network server.

Even so, QLToolkit II might not have been completed without the unrelenting encouragement from Hellmuth Stuen of QSOFTE, Denmark, whose indomitable faith in the technical merit of this product has kept me on my toes.

My thanks to you all.

Tony Tebby

QJUMP Toolkit II for the QL

Version II of the QJUMP Toolkit for the QL is an extended and improved version of the original QL Toolkit. This new version is largely rewritten to provide more facilities and to make the existing facilities of the QL and the QL Toolkit more powerful.

Since many of these improvements are to correct defects in the ROMs supplied with the QL, it would be better to supply an upgrade to the QL by replacing the Sinclair ROMs. Given the uncooperative attitude of Sinclair Research Limited towards such an upgrade, this Toolkit II is supplied as the next best thing.

1 Introduction

The Toolkit II attempts to put a large number of facilities into a consistent form. A little preamble is worthwhile to explain some of the principles.

This manual uses the following simple convention when describing commands and function calls:

CAPITAL LETTERS	are used for parts typed as is
<i>italic letters</i>	are used descriptively
lower case letters	are used as examples

Thus

VIEW name	is a description
VIEW fred	is an example

1.1 Commands Procedures Functions

The extensions to SuperBasic appear as extra commands, procedures and functions. The distinction between a command and a procedure is very slight and the two terms tend to be used interchangeably; the command is what a user types, the procedure is what does the work.

In some cases a command is used to invoke a procedure which in turn sets up and initiates a Job (e.g. **SPL** starts the resident spooler). A function is something that has a value and the name of a function cannot be used as a command: the value may be **PRINTed**, used in an expression or assigned to a variable.

1.2 Y/N/A/Q?

Y/N/A/Q? is a concise, if initially confusing, prompt that Toolkit II is bound to throw at the unsuspecting user from time to time. It is no more than a request for the user to press one of the keys Y (for yes), N (for no), A (for all), or Q (for Oh! Bother, I give up (Quit)).

What will actually happen when you press one of these keys, will depend on what you are trying to do at the time.

There is a short form which will only allow Y (for yes) and N (for no).

Before the reply to the Y/N/A/Q? (or Y or N?) prompt is read, any characters which have been typed ahead are discarded. Typing **BREAK (CTRL+space)** or **ESC** will have the same effect as a 'Q' (or 'N') keypress.

1.3 Overwriting

In some cases a command is given to create a new file with the name of a file which already exists. In general this will result not in an error message, but a prompt requesting permission to overwrite the file.

There are two (deliberate) exceptions to this rule:

OPEN_NEW will return an error, while the procedures **COPY_O**, **SAVE_O**, **SBYTES_O**, **SEXEC_O** and the spooler will happily overwrite their destination files without so much as a 'by your leave'.

1.4 #channel

All input and output from SuperBasic is through 'channels'. Some of these channels are implicit and are never seen (e.g. the command '**SAVE SER**' opens a channel to **SER**, lists the program to the channel, and closes the channel). Other are identified by a channel number which is a small, positive, integer preceded by a '#' (e.g. #2).

Many commands either allow or require a channel to be specified for input or output. This should be a SuperBasic channel number:

- #0 is the command channel (at the bottom of the screen),
- #1 is the normal output channel and
- #2 is the program listing channel.

Other channels (e.g. for communication with a file) may be opened using the SuperBasic **OPEN** commands (see section 10).

For interactive commands the default channel is #0, 'for most other commands the default channel is #1, for **LIST** and **ED** the default channel is #2, while for file access commands the default is #3.

For many of the commands it is possible to specify an implicit channel. This is in the form of 'X' followed by a file or device name. The effect of this is to open an implicit channel to the file or device, do the required operation and close the channel again.

E.g. DIR	list current directory to #1
DIR #2	list current directory to #2
DIR dlist	list current directory to file 'dlist'

this last example should be distinguished from

DIR dlist	list directory entries starting with dlist to #1
------------------	--

1.5 File and Device Names

In general it is possible to specify file or device names as either a normal SuperBasic name or as a string. The syntax of SuperBasic names limits the characters used in a name to letters digits and the underscore. There is no such limitation on characters used in a string. On a standard QL, a filename has to be given in full, but using the Toolkit II, the directory part of the name can be defaulted and just the filename used.

E.g. OPEN #3,fred	open file fred in the current directory
--------------------------	---

This gives rise to one problem: the SuperBasic interpreter has the unfortunate characteristic of trying to evaluate all the parameters of a command as expressions; in this example 'fred' will probably be an undefined variable which should not give rise to any problems. However, the command:

OPEN #3,list

will give an 'error in expression' error, as it is not possible for 'LIST', which is a command, to have a value. There are two ways round this problem: either avoid filenames which are the same as commands (procedures), functions or SuperBasic keywords (e.g., **FOR**, **END**, **IF** etc.), or put the name within quotes as a string:

OPEN #3,'list' or **OPEN #3,"list"**

1.6 CTRL F5

The **CTRL F5** keystroke (press **CTRL** and while holding it down press **F5**) is used to freeze the QL screen. Many commands in Toolkit II check their output window and, when it is full, internally generate a **CTRL F5** keystroke to hold the display until the user presses a key. (F5 will usually be the best key to press).

2 Contents of Toolkit II

SuperBasic is used as a command language on the QL as well as a programming language. Extensions are provided to improve the facilities of SuperBasic in both these areas as well as providing program development facilities.

The following list gives a comprehensive form of each command or function. There are often default values of the parameters to simplify the use of the procedures

2.1 Development Facilities

Section 3 File Editing

Toolkit II provides an editor and a command for viewing the contents of text files. ED is a window based editor for editing SuperBasic programs. VIEW is a command for examining line based files (e.g. assembler source files).

Commands

ED *#channel, line number* edit Superbasic program
 VIEW *#channel, name* view contents of a file

2.2 Command Language

The command language facilities of Toolkit II are intended to provide the QL with the control facilities to unlock the potential of the QDOS operating system. Most of these are 'direct' commands: they are typed in and acted on immediately. This does not mean that they may not be used in programs, but some care should be taken when doing this.

Section 4 Directory Control

QDOS does have a tree directory structure filing system! The Toolkit II provides a comprehensive set of facilities for controlling access to directories within this tree.

Commands

DATA__USE *name* set the default directory for data files
 PROG__USE *name* set the default directory for executable programs
 DEST__USE *name* set the default destination directory (COPY, WCOPY)
 SPL__USE *name* set the default destination device (SPL)
 DDOWN *name* move to a sub-directory
 DUP move up through the tree
 DNEXT *name* move to another directory at the same level

DLIST *#channel* list the defaults
Functions
 DATAD\$ function to find current data directory
 PROGD\$ function to find current program directory
 DESTD\$ function to find current default destination

Section 5 File Maintenance

All the filing system maintenance commands use the default (usually 'data') directories. Some of the commands are interactive and thus not suitable for use in SuperBasic programs: these are marked with an asterisk in this list. In these cases there are also simpler commands which may be used in programs. Depending on the command, the name given may be a generic (or 'wildcard') name referring to more than one file. With the exception of DIR (an extended version of the standard QL command DIR), all of these 'wildcard' commands have names starting with 'W'.

Commands

DIR *#channel, name* drive statistics and list of files
 WDIR *#channel, name* list of files
 STAT *#channel, name* drive statistics
 WSTAT *#channel, name* list of files and their statistics
 ASTAT *#channel, name* alphabetic list of files and their statistics

DELETE *name* delete a file
 *WDEL *#channel, name* delete files
 COPY *name TO name* copy a file
 COPY__O *name TO name* copy a file (overwriting)
 COPY__N *name TO name* copy a file (without header)
 COPY__H *name TO name* copy a file (with header)
 *WCOPY *#channel, name TO name* copy files
 SPL *name TO name* spool a file
 SPLF *name TO name* spool a file, <FF> at end
 RENAME *name TO name* rename a file
 *WREN *#channel, name TO name* rename files

Section 6 SuperBasic Programs

Toolkit II redefines and extends the file loading and saving operations of the QL. All the commands use the default directories. Additionally, the execution control commands have been extended to cater for the error handling functions of the 'JS' and 'MG' ROMs;

Commands

DO *name* do commands in file
 LOAD *name* load a SuperBasic program
 LRUN *name* load and run a SuperBasic program
 MERGE *name* merge a SuperBasic program
 MRUN *name* merge and run a SuperBasic program
 SAVE *name, ranges* Save a SuperBasic program
 SAVE__O *name, ranges* as SAVE but overwrites file if it exists
 RUN *line number* start a SuperBasic program
 STOP stop a SuperBasic program
 NEW reset SuperBasic
 CLEAR clear SuperBasic variables

Section 7 Load and Save

The binary load and save operations of the QL are extended to use the default directories.

Commands

LRESPR *name* load a file into resident procedure area and CALL
 LBYTES *name, address* load a file into memory at specified address
 CALL *address, parameters* CALL machine code with parameters
 SBYTES *name, address, size* save an area of memory
 SBYTES__O *name, address, size* as SBYTES but overwrites file if it exists
 SEXEC *name, address, size, data* save an area of memory as an executable file
 SEXEC__O *name, address, size, data* as SEXEC but overwrites file if it exists

Section 8 Program Execution

Program execution is, Anne Boleyn would be relieved to know, the opposite of program (ex)termination. The EXEC and EXEC__W commands in the standard QL are replaced by EX and EW in the QL Toolkit. Toolkit II redefines EXEC and EXEC__W to be the same as EX and EW. ET is for debuggers (no offence meant) only.

Commands

EXEC/EX program specifications load and set up one or more executable files

EXEC_W/EW program specifications ET program specification

Section 9 Job Control

The multitasking facilities of QDOS are made accessible by the job control commands and functions of Toolkit II.

Commands

JOBS #channel list current jobs

RJOB id or name, error code remove a job

SPJOB id or name, priority set job priority

AJOB id or name, priority activate a job

Functions

PJOB (id or name) find priority of job

OJOB (id or name) find owner of job

JOBS (id or name) find job name!

NXJOB (id or name, id) find next job in tree.

2.3 SuperBasic programming

Toolkit II has extensions to SuperBasic to assist in writing more powerful and flexible programs. The major improvements are in file handling and formatting.

Section 10 Open and Close

The standard QL channel OPEN commands are redefined by Toolkit II to use the data directory. In addition, Toolkit II provides a set of functions for opening files either using a specified channel number (as in the standard QL commands), or they will find and return a vacant channel number. The functions also allow filing system errors to be intercepted and processed by SuperBasic programs.

Commands

OPEN #channel, name open a file for read/write

OPEN_IN #channel, name open a file for input only

OPEN_NEW #channel, name open a new file

OPEN_OVER #channel, name open a new file, if it exists it is overwritten

OPEN_DIR #channel, name open a directory

CLOSE #channels close channels

Functions

FTST (name) test status of file

FOPEN (#channel, name) open a file for read/write

FOP_IN (#channel, name) open a file for input only

FOP_NEW (#channel, name) open a new file

FOP_OVER (#channel, name) open a new file, if it exists it is overwritten

FOP_DIR (channel, name) open a directory

Section 11 File Information

Toolkit II has a set of functions to read information from the header of a file.

FLEN (#channel) find file length

FTYP (#channel) find file type

FDAT (#channel) find file data space

FEXTRA (#channel) find file extra info

FNAME\$(#channel) find file name

FUPDT (#channel) find file update data

Section 12 Direct Access File

Toolkit has a set of commands for transferring data to and from any part of a file. The commands themselves

read or write 'raw' data, either in the form of individual bytes, or in SuperBasic internal format (integer, floating point or string).

Commands

BGET #channel\position, items get bytes from a file

BPUT #channel\position, items put bytes onto a file

GET #channel\position, items get internal format data from a file

PUT #channel\position, items put internal format data onto a file

TRUNCATE #channel\position truncate file

FLUSH #channel flush file buffers

Functions

FPOS (#channel) find file position

Section 13 Format Conversions

Toolkit II provides a number of facilities for fixed format I/O. These include binary and hexadecimal conversions as well as fixed format decimal.

Commands

PRINT_USING #channel, format, fixed format output list of items to print

Functions

FDEC\$ (value, field, ndp) fixed format decimal

IDEC\$ (value, field, ndp) scaled fixed format

CDEC\$ (value, field, ndp) decimal

FEXP\$ (value, field, ndp) fixed exponent format

HEX\$ (value, number of bits) convert to hexadecimal

BIN\$ (value, number of bits) convert to binary

HEX (hexadecimal string) hexadecimal to value

BIN (binary string) binary to value

Section 14 Display Control

Toolkit II provides commands for enabling and disabling the cursor as well as setting the character font and sizes or restoring the windows to their turn on state.

Commands

CURSEN #channel enable the cursor

CURDIS #channel disable the cursor

CHAR_USE #channel, addr1, addr2 set or reset the character font

CHAR INC #channel, x inc, y inc set the character x and y increments

WMON mode reset to 'Monitor'

WTV mode reset to 'TV' windows

Section 15 Memory Management

Toolkit II has a set of commands and functions to provide memory management facilities within the 'common heap' area of the QL.

Functions

FREE_MEM find the amount of free memory

ALCHP (number of bytes) allocates space in common heap (returns the base address of the space)

Commands

RECHP base address return space to common heap

CLCHP clear out all allocations in the common heap

DEL_DEF6 delete file definition blocks from common heap

Section 16 Procedure Parameters

Four functions are provided by Toolkit II to improve the handling of procedure (and function) parameters. Using these it is possible to determine the type (integer, floating point or string) and usage (single value or array) of the calling parameter as well as the 'name'.

PARTYP (*name*) find type of parameter
PARUSE (*name*) find usage of parameter
PARNAM\$ (*parameter number*) find name of parameter
PARSTR\$ (*name, parameter number*) if parameter 'name' is a string, find the value, else find the string.

Section 17 Error Handling

These facilities are provided for error processing in version JS and MG of SuperBasic.

ERR_DF true if drive full error has occurred
REPORT ___#*channel, error number* report an error
CONTINUE *line number* continue or retry from a specified line

Section 18 Time-keeping

Two clocks are provided in Toolkit II, one configurable digital clock, and an alarm clock.

CLOCK #*channel, format* variable format clock
ALRM *hours, minutes* alarm clock

Section 19 Extras

EXTRAS lists the extra facilities linked into SuperBasic
TK2_EXT enforces the Toolkit II definitions of common commands and functions

2.4 Extensions to Drive

In addition to the SuperBasic interpreter, Toolkit II has important extensions to the console, Microdrive and Network device drivers.

3. File Editing

3.1 ED - SuperBasic Editor

ED is a small editor for SuperBasic programs which are already loaded into the QL. If the facilities look rather simple and limited, please remember that the main design requirement of ED is the small size to leave room for other facilities.

ED is invoked by typing:

ED
 or ED *line number*
 or ED #*channel number*
 or ED #*channel number, line number*

If no line number is given, the first part of the program is listed, otherwise the listing in the window will start at or after the given line number. If no channel number is given, the listing will appear in the normal SuperBasic edit window #2. If a window is given, then it must be a CONSOLE window, otherwise a 'bad parameter' error will be returned. The editor will use the current ink and paper colours for normal listing, while using white ink on black paper (or vice versa if the paper is already black or blue) for 'highlighting'. Please avoid using window #10 for the ED.

The editor makes full use of its window. Within its window, it attempts to display complete lines. If these lines are too long to fit within the width of the window,

Section 20 Console Driver

Toolkit II provides last line recall for the command #0 as well as allowing strings of characters to be assigned to 'ALT' keystrokes received on this channel.

Also, for MG versions ROMs only, it provides a patch to correct the POINT, short LINE and SHORT ARC problems in the MG ROMs.

Commands

<ALT> <ENTER> keystroke recovers last line typed
ALTKEY *character, strings* assign a string to <ALT> character keystroke

Section 21 Microdrive Driver

Toolkit II extends the microdrive driver to provide OPEN file with overwrite, as well as TRUNCATE and RENAME of files. These facilities are supported at QDOS level (Traps #2 and #3) as well as from SuperBasic. The FLUSH operation is respesified to set the file header as well as flush the buffers.

Section 22 Network Driver

The network driver is enhanced to provide a primitive form of broadcast communication as well as providing a comprehensive file server program which allow many QLs to share a disc system or printer.

Commands

FSERVER invokes the 'file server'
NFS_USE *name, network names* sets the network file server name

Device names

Station number ___ *IO device* the name of a remote IO device (e.g. N2_FLP1
 is floppy 1
 on network station 2)

they are 'wrapped around' to the next row in the window: these extra rows are indented to make this 'wrap around' clear. For ease of use, however, the widest possible window should be used.

ED must not be called from within a SuperBasic program.

The ESC key is used to return to the SuperBasic command mode.

After ED is invoked, the cursor in the edit window may be removed using the arrow keys to select the line to be changed. In addition the up and down keys may be used with the ALT key (press the ALT key and while holding it down, press the up or down key) to scroll the window while keeping the cursor in the same place, and the up and down keys may be used with the SHIFT key to scroll through the program a 'page' at a time.

The editor has two modes of operation: insert and overwrite. (To change press F4.) There is no difference between the modes when adding characters to or deleting characters from the end of a line. Within a line, however, insert mode implies that the right hand end of a line will be moved to the right when a character is inserted, and to the left when a character is deleted. No part of the line is moved in overwrite mode. Trailing spaces at the end of a line are removed automatically.

To insert a new line anywhere in the program, press ENTER. If there is no room between the line the cursor

is on and the next line in the program (e.g. the cursor is on line 100 and the next line is 101) then the ENTER key will be ignored, otherwise a space is opened up below the current line, and a new line number is generated. If there is a difference of 20 or more between the current line number and the next line number, the new line number will be 10 on from the current line number, otherwise, the new line number will be half way between them.

If a change is made to a line, the line is highlighted: this indicates that the line has been extracted from the program. The editor will only replace the line in the program when ENTER is pressed, the cursor is moved away from the line, or the window is scrolled. If the line is acceptable to SuperBasic, it is rewritten without highlighting. If, however, there are syntax errors, the message 'bad line' is sent to window #0, and the line remains highlighted.

While a line is highlighted, ESC may be used to restore the original copy of the line, ignoring all changes made to that line.

If a line number is changed, the old line remains and the new line is inserted in the correct place in the program. This can be used to copy single lines from one part of the program to another.

If all the visible characters in a line are deleted, or if all but the line number is deleted, then the line will be deleted from the program. An easier way to delete a line is to press CTRL and ALT and then the left arrow as well. The length of lines is limited to about 32766 bytes. Any attempt to edit longer lines may cause undesirable side effects. If the length of a line is increased when it is changed, there may be a brief pause while SuperBasic moves its working space.

3.2 Summary of Edit Operations

The general usage of the keys follows the Concepts section of the QL User Guide first, and then the business programs usage.

TAB	tab right (columns of 8)
SHIFT TAB	tab left (columns of 8)
ENTER	accept line and create a new line
ESC	escape - undo changes or return to SuperBasic
up arrow	move cursor up a line
down arrow	move cursor down a line
ALT up arrow	scroll up a line (the screen moves down!)
ALT down arrow	scroll down a line (the screen moves up!)
SHIFT up arrow	scroll up one page
SHIFT down arrow	scroll down one page
left arrow	move cursor left one character
right arrow	move cursor right one character
CTRL left arrow	delete one character to left of cursor
CTRL right arrow	delete character under cursor
CTRL ALT left arrow	delete line
SHIFT F4	change between overwrite and insert mode

3.3 Viewing a file

VIEW is procedure intended to allow a file to be examined in a window on the QL display. The default window is #1.

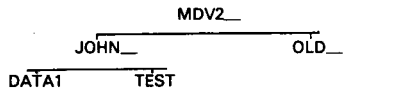
View is invoked by typing

- VIEW name** View file 'name' in window #1
 - VIEW #channel, name** View file 'name' in given window
 - VIEW name 1, name2** Send file 'name2' to 'name1'
- VIEW** truncates lines to fit the width of the window. When the window is full, CTRL F5 is generated. If the output device (or file) is not a console, then lines are truncated to 80 characters.

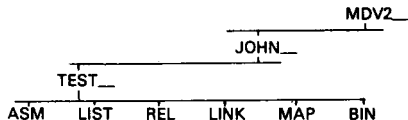
4. Directory Control

4.1 Directory Structures

In QDOS terminology, a 'directory' is where the systems expects to find a file. This can be as simple as the name of a device (e.g. MDV2_ the name of the Microdrive number 2) or be much more complex forming part of a 'directory tree' (directories grow on trees - honestly they do). For example: the directory MDV2_ could include directories JOHN_ and OLD_ (note: all directory names end with an '_'), and JOHN_ could include files DATA1 and TEST.



This can be illustrated with the case of an assembler program TEST, processed using the GST macro assembler and linkage editor. The assembler source file (TEST_ASM), the listing output from the assembler (TEST_LIST), the relocatable output from the assembler (TEST_REL), the linker control file (TEST_LINK), the linker listing output (TEST_MAP) and the executable program produced by the linker (TEST_BIN) are all treated as files within the directory TEST_.



This shows another characteristic of the 'directory tree': it grows downwards. The complete QDOS filename for DATA1 in this example is MDV2_ JOHN_ DATA1. (You may have come across the terms 'pathname' or 'treename' in other operating systems; these refer to the same thing as a QDOS filename).

One unusual characteristic of the QDOS directory structure is the absence of a formal file name 'extension'. This is not strictly necessary as 'extensions' (e.g. _aba for ABACUS files, _asm for assembler source files etc.) are treated as files within a directory.

Thus Toolkit II provides facilities to set default directories. The defaults are available for all filing system operations. A default may be set to any level of complexity and gives a starting point for finding a file in the tree structure. Thus, in this example, if the default is MDV2_, then JOHN_TEST_ASM will find the assembler source. If the default is MDV2_ JOHN_, then TEST_ASM will find it, while the full filename MDV2_ JOHN_TEST_ASM will find the file regardless of the default.

4.2 Setting Defaults

Unusually, the Toolkit II extensions to QDOS support three distinct defaults for the directory structure. This is because QDOS is an intrinsically multi-drive operating system. It is expected that executable programs will be in a different directory, and probably on a different drive, from any data files being manipulated.

Furthermore, the copying procedure are more likely to be used to copy from one directory to another, or from the filing system to a printer or other output device, than they are to be used to copy files within a directory.

There are three commands for setting the three defaults:

```
DATA__USE directory name      set data default
PROG__USE directory name      set program default
DEST__USE directory name      set destination default
```

If the directory name supplied does not end with '___', '___' will be appended to the directory name.

The **DATA__USE** default is used for most filing system commands in the Toolkit. The **PROG__USE** default is used only for finding the program files for the EX/EXEC commands; whilst the **DEST__USE** default is used to find the destination filename when the file copying and renaming commands (**SPL**, **COPY**, **RENAME** etc.) are used with only one filename.

There is a special form of **DEST__USE** command which does not append '___' to the name given. This provides the default destination device for the spooler:

SPL__USE *device name*

This sets the destination default, but if there is no '___' at the end, it is not treated as a directory and so, if a destination filename is required, the default will be used unmodified.

E.g. **DEST__USE flp__old** (default is **FLP2__OLD__**)

```
.....
SPL fred
or SPL__USE flp2__old_(default is FLP2__OLD__)
.....
SPL fred
```

Both of these examples will spool **FRED** to **FLP2__OLD__FRED**. Whereas if **SPL__USE** is used with a name without a trailing '___' (i.e. not a directory name) as follows

```
SLP__USE ser      (default is SER)
.....
SPL fred
```

then **FRED** will be spooled to **SER** (not **SER__FRED**).

Note that **SPL__USE** overwrites the **DEST__USE** default and vice versa.

4.3. Directory Navigation

Three commands are provided to move through a directory tree.

```
DDOWN name      move down (append 'name' to the
                  default)
DUP Move up (strip off the last level of the directory)
DNEXT name      move up and then down a different
                  branch of the tree
```

It is not possible to move up beyond the drive name using the **DUP** command. At no time is the default name length allowed to exceed 32 characters.

These commands operate on the data default directory. Under certain conditions they may operate on the other defaults as well:

If the program default is the same as the data default, then the two defaults are linked and these commands will operate on the **PROG__USE** default as well. If the destination default ends with 'O' (i.e. it is a default directory rather than a default device), then these commands will operate on the destination default.

These rules are best seen in action.

initial values	data	program	destination
DDOWN john	mdv2__	mdv1__	ser
DNEXT fred	mdv2__fred__	mdv1__	ser
PROG__USE mdv2__fred	mdv2__fred__	mdv2__fred__	ser
DUP	mdv2__john__	mdv2__john__	ser
DEST__USE mdv1	mdv2__	mdv2__	mdv1__
DDOWN john	mdv2__john__	mdv2__john__	mdv1__john__
SPL__USE serlc	mdv2__john__	mdv2__john__	serlc

4.4 Taking Bearings

Should you wonder where you are in the directory tree, there is a command list all three defaults:

DLIST list data, program and destination
or **DLIST #channel** defaults
or **DLIST \name'** defaults

if an output channel is not given, the defaults are listed in **window #1**.

To find the defaults from within a SuperBasic program there are three functions:

```
DATAD$ find the data default
PROGD$ find the program default
DESTD$ find the destination default
```

The functions to find the individual defaults should be used without any parameters. E.g.

IF DATAD\$ <> PROGD\$: PRINT 'Separate directories'

DEST\$ = DESTD\$

IF DEST\$ (LEN (DEST\$)) = '___':

PRINT 'Destination'1 DEST\$

Facilities to enable executable programs to find the default directories were provided in the original Sinclair QL Toolkit, and the same facilities are provided in this Toolkit. These facilities are not widely used in commercial software for the QL.

The real solution of providing the default directories at QDOS trap level can only be attained using additional hardware in the expansion slot or by replacement operating system ROMs. You will probably find, therefore, that much commercially written software will not recognise the defaults you have set. There is an example of overcoming this problem in the example program appendix A.

5 File Maintenance

The standard file maintenance procedure of the QL (COPY, DELETE and DIR) are filled out into a comprehensive set in Toolkit II. All of the commands, both standard and new, use the directory defaults; in addition, many of the commands use wild card names to refer to groups of similarly named files.

5.1 Wild Card Names

A wild card name is a special type of filename where part of the name is treated as a 'wild card' which can be substituted by any string of characters. If, for convenience, the wild card name is to be a normal SuperBasic name, then special characters cannot be used for the wild card (e.g. myfiles_*.asm would be treated by SuperBasic as an arithmetic expression and SuperBasic would attempt to multiply myfiles_ by *.asm).

For this reason a simpler scheme is adopted: any missing section of a file name is treated as a wild card. The end of a wild card name is implicitly missing.

If the wild card name is not a full file name, the default directory is added to the start of the name.

In the following example, the default directory is assumed to be FLP2_.

Wild card name	Full wild card name	Typical matching files
fred	flp2_fred	flp2_fred flp2_freda_list
__fred	flp2_fred	flp2_fred flp2_freda_list flp2_old_fred flp2_old_fred_list
flp1_old_list	flp1_old_list	flp1_old_jo_list flp1_old_fred_list

5.2 Directory Listing

There are two forms of directory listing: the first lists just the filenames, the second lists the filenames together with file size and update date. All the commands use wild card names and the data default directory. The output from these commands will be sent to channel #1 by default; but a channel or implicit channel may be specified: if the output channel is to a window the listing is halted (CTRL F5) when the window is full.

DIR #channel, name drive statistics and list of files

WDIR #channel, name list of files

WSTAT #channel, name list of files and their statistics

In all cases the channel specification and the name are optional.

The possible forms of (for example) **WDIR** are

WDIR list current directory to #1

or **WDIR #channel** list current directory to #channel

or **WDIR \name** list current directory to 'name'

or **WDIR name** list directory 'name' to #1

or **WDIR #channel, name** list directory 'name' to #channel

or **WDIR \name1, name2** list directory 'name2' to 'name1'

E.g.

WDIR \ser, _asm list all _asm files in current directory to SER

WDIR flp1_ list all files on FLP1_ in window #1

WDIR #3

list all files in current directory to channel #3

DIR is provided for compatibility only: before listing the files, the drive statistics (medium name, number of vacant sectors/number of good sectors) are written out.

5.3 Drive Statistics

There is one command to print the statistics for the drive holding a specified directory, or the data default directory.

STAT #channel, name
or **STAT \name1, name2**

Both the channel and the name are optional.

5.4 File Deletion

The standard procedure **DELETE** has been modified to use the data default directory unless a full file name is supplied. No error is generated if the file is not found. There are also two interactive commands to delete many files using wild card names.

DELETE name delete one file

WDEL #channel, name delete files

For **WDEL** both the channel and the name are optional.

E.g.

WDEL delete files from current directory

WDEL__list delete all __list files from current directory

Unless a channel is specified, the wild card deletion procedures use the command **WINDOW #0** to request confirmation of deletion. There are four possible replies:

Y (yes) delete this file

N (no) do not delete this file

A (all) delete this and all the next matching files

Q (quit) do not delete this or any of the next files

5.5 File Copying

The two forms of the **COPY** command provided with the QL are changed to use default filenames, and also to provide more flexibility. A number of other commands are added.

Files in QDOS have headers which provide useful information about the file that follows. It depends on the circumstances whether it is a good idea to copy the header of the file when the file is copied.

It is a good idea to copy the header when:-

a) copying an executable program file so that the additional file information is preserved,

b) copying a file over a pure byte serial link so that the communications software will know in advance the length of the file.

It is a bad idea to copy the header when:

c) copying a text file to a printer because the header will be likely to have control codes and spurious or unprintable characters.

The general rules used by the **COPY** procedures in Toolkit II, are that the header is only copied if there is additional information in the header. This caters for cases (a) and (c) above. A **COPY__N** command is included for compatibility with the standard **QL COPY__N**: this never copies the header. A **COPY__H** command is included to copy a file with the header to cater for case (b) above. (Note that the standard **QL** command **COPY** always copies the header.) Neither **COPY__N** nor **COPY__H** need ever be used for file to file copying.

A second rule used by the COPY (as well as by the WREN) procedures is that if the destination file already exists, then the user will be asked to confirm that overwriting the old file is acceptable. The COPY_O (copy overwrite) and the spooler procedures do not extend this courtesy to the user.

If the commands are given with two filenames then the data default directory is used for both files. If, however, only one filename (or, in the case of the wild card procedures, no name at all) is given then the destination will be derived from the destination default.

a) if the destination default is a directory (ending with '_'), set by DEST_USE) then the destination file is the destination default followed by the name,

b) if the destination default is a device (not ending with '_'), set by SPL_USE) then the destination is the destination default unmodified.

5.5.1 Single File Copies

COPY name TO name copy a file
COPY_name TO name copy a file (overwriting)
COPY__name TO name copy a file (without header)
COPY_H name TO name copy a file (with header)

These commands can be given with one or two names. The separator 'TO' is used for clarity, but you may use a comma instead.

To illustrate the use of the copy command, assume that the data default is MDV2_ and the destination default is MDV1_.

COPY fred TO old_fred copies mdv2_fred to mdc2_old_fred

COPY fred, ser copies mdv2_fred to ser

COPY fred copies mdv2_fred to mdv1_fred

SPL_USE ser . . .

COPY fred COPIES MDV2_fred to ser

5.5.2 Wild Card Copies

The interactive copying procedure WCOPY is used for copying all or selected parts of directories. The command may be given with both source and destination wild card names, with one wild card name or with no wild card names at all. Giving the command with no wild card names has the same effect as giving one null name:

WCOPY and WCOPY'' are the same

If you get confused by the following rules about the derivation of the copy destination, just use WCOPY intuitively and look carefully at the prompts.

If the destination is not the destination default device, then the actual destination file name for each copy operation is made up from the actual source file name and the destination wild name. If a missing section of the source wild name is matched by a missing section of the destination wild name, then that part of the actual source file name will be used as the corresponding part of the actual destination name. Otherwise the actual destination file name is taken from the destination wild name. If there are more sections in the destination wild name than in the source wild name, then these extra sections will be inserted after the drive name, and vice versa.

The full form of the command is:

WCOPY #channel, name TO name copy files

The separator TO is used for clarity, you may use a comma instead.

If the channel is not given (i.e. most of the time), then the requests for confirmation will be sent to the

command channel #0, to the chosen channel, and the user is requested to press one of:

Y (yes) copy this file
N (no) do not copy this file
A (all) copy this and all the next matching files
Q (quit) do not copy this or any other files

If the destination file already exists, the user is requested to press one of:

Y (yes) copy this file, overwriting the old file
N (no) do not copy this file
A (all) overwrite the old file, and overwrite any other files requested to be copied
Q (quit) do not copy this or any other files

For example, if the default directory is flp2_, and the default destination is flp1_

WCOPY would copy all files on flp2_ to flp1_

WCOPY flp1__flp2__ would copy all files on flp1_ to flp2_

WCOPY fred would copy flp2_fred to flp1_fred
 flp2_freda_list to flp1_freda_list

WCOPY fred,mog would copy flp2_fred to flp2_mog
 flp2_freda_list to flp1_moga_list

WCOPY__fred,__mog would copy flp2_fred to flp2_mog
 flp2_freda_list to flp2_moga_list

flp2_old_freda_list to flp2_old_moga_list

WCOPY __list,old__ would copy flp2_jo_list to flp2_old_jo_list
 flp2_freda_list to flp2_old_freda_list

WCOPY old_list,flp1__ would copy flp2_old_jo_list to flp1_jo_list
 flp2_old_freda_list to flp1_freda_list

5.5.3 Background Copying

A background file spooler is provided which copies files in the same way as COPY_O (Section 5.5.1), but is primarily intended for copying files to a printer. As an option, a form feed (ASCII <FF> decimal 12, hex 0C) can be sent to the printer at the end of the file.

SPL name TO name spool a file

SPLF name TO name spool a file, <FF> at end

The normal use of this command is with one name only.

SPL_USE ser2 set spooler default TO serial Port 2

.....

SPLF fred spool fred to ser2, adding a form feed to the file

When used in this way, if the default device is in use, the Job will be suspended until the device is available. This means that many files can be spooled to a printer at once.

A variation on the SPL and SPLF commands is to use SuperBasic channels in place of the filenames. These channels should be opened before the spooler is invoked:

SPL #channel1 TO #channel2

Where channel1 must have been opened for input and channel2 must have been open for output.

5.5.4 Renaming Files

Renaming a file is a process similar to COPYING a file, but the file itself is neither moved nor duplicated, only the directory name is changed. The commands, however, are exactly the same in use as the equivalent COPY commands.

RENAME name TO name see COPY

WREN #channel, name TO name see WCOPY

6 SuperBasic Programs

All the commands for loading, saving and running SuperBasic programs have been redefined in Toolkit II. The differences are in the areas of:

- a) default filenames,
- b) **WHEN ERROR** (JS and MG ROMs only),
- c) common heap handling.

6.1 DO

There is one additional procedure, **DO**, to execute SuperBasic commands from file.

DO name do commands in the file
 e.g.: the contents of file "set printer" could be:

```
OPEN #3,ser1: PRINT #3, CHR$(27); "C"; "H";
CLOSE #3
```

Set form length to European standard 72 lines per page on an EPSON/Sinclair compatible printer.

If we assume that the file "set printer" is stored on the "current directory" and default "data" device, you can set your printer, just by saying:

DO set printer

The commands should be of the 'direct' type: any lines with line numbers will be merged into the current SuperBasic program. The file should not contain any of the commands listed in this section (e.g. **RUN**, **LOAD** etc.). **CONTINUE**, **RETRY** or **GOTO**. It appears that a **DO** file can invoke SuperBasic procedures without harmful effect.

A **DO** file can contain in line clauses:

```
FOR i = 1 TO 20: PRINT 'This is a DO file'
```

If you try to **RUN** a Basic program from a **DO** file, then the file will be left open. Likewise, if you put direct commands in a file that is **MERGED**, then the file will be left open.

6.2 Default Directories

Most of the commands use the data default directory. In addition, the program **LOADing** commands will try the program default directory if a file cannot be found in the data default directory.

6.3 WHEN ERROR Problems

There is a problem in the JS and MG ROM error handling code, in that **WHEN ERROR** processing, once set, is never reset, even if the **WHEN ERROR** clause is removed by a **NEW** or a **LOAD**! All of the commands in this section clear the **WHEN ERROR** processing flag, and all but **STOP** also clear the pointer to the current **WHEN ERROR** clause.

6.4 Common Heap

Toolkit II contains facilities for allocating space in the common heap. This space is cleared by the commands that clear the SuperBasic variables: **LOAD**, **LRUN**, **NEW** and **CLEAR**.

6.5 Summary of Commands

DO name	do commands in the file.
LOAD name	load a SuperBasic program.
LRUN name	load and run a SuperBasic program.
MERGE name	merge a SuperBasic program
MRUN name	merge and run a SuperBasic program
SAVE name, ranges	save a SuperBasic program
SAVE_O name, ranges	as SAVE but overwrites the file if it exists
RUN line number	start a SuperBasic program.
STOP	stop a SuperBasic program.
NEW	reset SuperBasic
CLEAR	clear SuperBasic variables

7 Load and Save

Toolkit II provides the same binary file load and save operations as the standard QL. The differences are that the save operations will request permission to overwrite if the file already exists, and all the commands use default directories.

There are also two 'overwrite' variants for the save operations, and one new command: **LRESPR**.

LRESPR opens the load file and finds the length of the file, then reserves space for the file in the resident procedure area before loading the file. Finally a **CALL** is made to the start of the file.

The **CALL** procedure itself has been overwritten to avoid the problems that occur in **AH** and **JM ROMs** when a **CALL** is made from a large (32 bytes) program.

LRESPR name load a file into resident procedure area and **CALL**
LBYTES name, address load a file into memory at specified address

CALL address, parameters **CALL** machine code with parameters
SBYTES name, address, size save an area of memory
SBYTES_O name, address, size as **SBYTES** but overwrites file if it exists
SEXEC name, address, size, data save an area of memory as an executable file
SEXEC_O name, address, size, data as **SEXEC** but overwrites

For **SEXEC** and **SEXEC_O** the 'data' parameter is the default data space required by the program.

If there are any Jobs in the QL (apart from Job 0 the SuperBasic interpreter) then **LRESPR** will fail with the error message "not complete". If this happens, use **RJOB** to remove all the other jobs.

8 Program Execution

There is one procedure of initiating the execution of compiled (executable) programs. This procedure is invoked by five commands: EX, EXEC (which are synonymous), EW, EXEC_W (which are synonymous) and ET. The differences are very small: when EX is complete, it returns to SuperBasic; when EW is complete it waits until the programs initiated have finished before returning to SuperBasic; while ET sets up the programs, but returns to SuperBasic so that a debugger can be called to trace the execution. EX will be used to describe all the commands.

8.1 Single Program Execution

In its simplest form EX can be used to initiate a single program:

EX name

The program in the file 'name' is loaded into the transient program area of the QL and execution is initiated. If the file does not contain an executable program, a 'bad parameter' error is returned.

It is also possible to pass parameters to a program in the form of a string:

EX name; parameter string

In this case the program in the file 'name' is loaded into the transient program area, the string is pushed onto its stack and execution is initiated.

Finally it is possible for EX to open input and output files for a program as well as (or instead of) passing its parameters. If preferred, a SuperBasic channel number may be used instead of a filename. A channel used in this way must already be open.

EX program name, file names or #channel; parameter string

Taking as an example the program UC which converts a text file to upper case, the command:

```
EX uc, fred, #1
```

will load and initiate the program UC, with fred as its input file and the output being sent to window #1.

8.2 Filters

EX is designed to set up filters for processing streams of data.

Within the QL it is possible to have a chain of cooperating jobs engaged in processing the same data in a form of production line. When using a production line of this type, each job performs a well-defined part of the total process. The first job takes the original data and does its part of the process; the partially processed data is then passed on to the next job which carries out its own part of the process; and so the data gradually passes through all the processes. The data is passed from one Job to the next through a 'pipe'. The data itself is termed a 'stream' and the Jobs processing data are termed 'filters'.

9 Job Control

As QDOS is a multitasking operating system, it is possible to have a number of competing or co-operating jobs in the QL at any one time. Jobs 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 priority and its position within the tree of Jobs (ownership). A Job is identified by two numbers: one is the Job number

Using the symbols[] to represent a single optional item {} to represent a repeated optional item the complete form of the EX command is:

```
EX[ #channelTO" ]prog_spec  
{ TO prog_spec } [TO #channel"]
```

where prog_spec is

program name

```
{ ,file name or #channel } [,parameter string"]
```

Each TO separator creates a pipe between jobs

All the names and the parameter string may be names, strings or string expressions. The significance of the filenames is, to some extent, program dependent; but there are two general rules which should be used by all filters:

- 1) the primary input of a filter is the pipe from the previous Job in the chain (if it exists), or else the first data file,
- 2) the primary output of a filter is the pipe to the next job in the chain (if it exists) or else the last data file.

Many filters will have only two IN/OUT channels: the primary input and the primary output.

If the parameters of EX start with '#channel TO', then the corresponding SuperBasic channel will be closed (if it was already open) and a new channel opened as a pipe to the first program;

Any data sent to this channel (e.g. by PRINTing to it) will be processed by the chain of Jobs. When the channel is CLOSED, the chain of Jobs will be removed from the QL.

If the parameters of EX end with 'TO #channel', then the corresponding SuperBasic channel will be closed (if it was already open) and a new channel opened as a pipe from the last program.

Any data passing through the chain of Jobs will arrive in this channel and may be read (e.g. by INPUTing from it). When all the data has passed, the Jobs will remove themselves and any further attempt to take input from this channel will get an 'end of file' error. The EOF function may be used to test for this.

8.3 Example of Filter Processing

As an example of filter processing, the programs UC to convert a file to upper case, LNO to line number a file, and PAGE to split a file onto pages with an optional heading are all chained to process a single file:

```
EX uc, fred TO into TO page,ser; 'File fred at "'& date$
```

The filter UC takes the file 'fred' and after converting it to upper case, passes through a pipe to LNO. LNO adds line numbers to each line and passes the file down a pipe to OAGE. In its turn, PAGE splits the file onto pages with the heading (including in this case the date) at the top of each page, before sending the file to the SER port. Note that the file fred itself is not modified; the modified versions are purely transient.

which is an index into the table of Jobs, and the other is a tag which is used to identify a particular Job so that it cannot be confused with a previous Job occupying the same position in the Job table. With QDOS the two numbers are combined into the Job ID: Thus: JOBID = Job number + tag*65536. For these Job control routines, where Job_id is a parameter of one of the Job control routines, it may be given as either a single number (the Job ID, as returned from OJob or

NXJOB of Toolkit II) or as a pair of numbers (Job number, Jobtag). Thus the single parameter 85538 (2+1*65536) is equivalent to the two parameters 2,1.

9.1 Job Control Commands

JOBS is a command to list all the Jobs running in the QL at the time. If there are more Jobs in the machine than can be listed in the output window, the procedure will freeze the screen (CTRL F5) when it is full. The procedure may fail if Jobs are removed from the QL while the procedure is listing them. The following information is given for each Job:

- the Job number
- the Job tag
- the Job's owner Job number
- a flag 'S' is the Job is suspended
- the Job priority
- the Job (or program) name.

The command is

JOBS list current Jobs to #1
JOBS #channel list current Jobs
JOBS name list Jobs to 'name'

There are three procedures for controlling Jobs in the QL:

RJOB id or name, error code remove a Job
OJOB id or name, priority set Job priority
AJOB id or name, priority activate a Job

If a name is given rather than a Job ID, then the procedure will search for the first Job it can find with the given name.

If there is a Job waiting for the completion of a Job removed by **RJOB**, it will be released with **D0** set to the error code.

E.g.
RJOB 3,8,-1 remove Job 3, tag 8 with error -1
SPJOB demon,1 set the priority of the Job called 'demon' to 1

9.2 Job Status Functions

The Job status functions are provided to enable a SuperBasic program to scan the Job tree and carry out complex Job control procedures.

PJOB (id or name) find priority of Job
OJOB (id or name) find owner of Job
JOBS (id or name) find Job name
NXJOB (id or name) top Job id) find next Job in tree

NXJOB is a rather complex function. The first parameter is the id of the Job currently being examined, the second is the id of the Job at the top of the tree. If the first id passed to **NXJOB** is the last Job owned, directly or indirectly, by the 'top Job', then **NXJOB** will return the value 0, otherwise it will return the id of the next Job in the tree.

Job 0 always exists and owns directly or indirectly all other Jobs on the QL. Thus a scan starting with id = 0 and top Job id = 0 will scan all Jobs in the QL.

It is possible that, during a scan of the tree, a Job may terminate. As a precaution against this happening, the Job status functions return the following values if called with an invalid Job id:

PJOB=0 OJOB=0 JOBS="" NXJOB=-1

10 Open and Close

All of **OPEN** and **CLOSE** commands and functions avoid the problem that occurs using the standard QL facilities when more than 32768 files have been opened in one session.

10.1 Open Commands

The **OPEN** commands of the standard QL have been modified to use the data default directory. Two commands have been added to open a new file overwriting the old file if it already exists, and to open a directory.

OPEN
OPEN__IN #channel, name open a file for input only
OPEN__NEW #channel, name open a new file
OPEN__OVER #channel, name open a new file, if it exists it is overwritten
OPEN__DIR #channel, name open a directory

10.2 File Status

The function **FTEST** is used to determine the status of a file or device. It opens a file for input only and immediately closes it. If the file exists it will either return the value 0 or -9 (in use error code), if it does not exist, it will return -7 (not found error code). Other possible returns are -11 (bad name), -15 (bad parameter), -3 (out of memory) or -6 (no room in the channel table).

FTEST (name) test status of file
 The function can be used to check that a file does not exist:

IF FTEST (file\$) xx -7: PRINT 'file'; file\$; 'exists'

10.3 File Open Functions

This is a set of functions for opening files. These functions differ from the **OPEN** procedures 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 may be used to find a vacant hole in the channel table: if successful they return the channel number.

FOPEN (#channel, name) open a file for read/write
FOP__IN (#channel, name) open a file for input only
FOP__NEW (#channel, name) open a new file
FOP__OVER (#channel, name) open a new file, if it exists it is overwritten
FOP__DIR (#channel, name) open a directory

When called with two parameters, these functions return the value zero for successful completion, or a negative error code.

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

ferr = FOP__IN (#3,name\$&'__ASM') :REMark try to open__ASM file

IF ferr = -7 :ferr = FOP__IN (#3,name\$) :REMark ERR.NF, try no__ASM

The **#channel** parameter is optional: if it is not given, the functions will search the channel table for a vacant entry, and, if the open is successful, the channel number will be returned. Note that error codes are always negative, and channel numbers are positive.

In this example:

```

outch = FOP__NEW (fred)           :REMark open fred
if outch <0:REPORT outch:STOP     :REMark ... oops
PRINT #outch, 'This is file Fred'
CLOSE #outch
there is no need to ever know the actual channel
number.
    
```

10.4 CLOSE

The CLOSE command has been extended to take multiple parameters. In addition, if called with no parameters it will close all channel numbers #3 and above. It will not report an error if a channel is not open.

```

CLOSE #channels                    close channels
E.g. CLOSE #3, #4, #7             close #3, #4 and #7
    
```

11 File Information

There are six functions to extract information from the header of the file.

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 the file by the command GET #n\,999999.)

```

FLEN (#channel)                   find file length
FTYP (#channel)                   find file type
FDAT (#channel)                   find file data space
FXTRA (#channel)                  find file extra info
    
```

```

FNAME$ (#channel)                 find filename
FUPDT (#channel)                 find file update date
The file type is                 0 for ordinary files
                                1 for executable programs
                                2 for relocatable machine code.
    
```

The file information functions can also be used with implicit channels. E.g.

```

PRINT FLEN (#3)                  print the length of the file open
                                on channel #3
PRINT FLEN (\,fred)             print the length of file fred
    
```

12 Direct Access Files

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, IN/OUT, 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.

To keep files tidy there is a command to truncate a file (when information at the end of a file is no longer required), and a command to flush the file buffers.

A direct access input or output (I/O) command specifies the I/O channel, a pointer to the position in the file for the I/O operation to start and a list of items to be input or output.

command #channel\position, items

It is usual (although not essential - the default is #3) to give a channel number for the direct I/O commands. If no pointer is given, the routines will read or write from the current position, otherwise 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 new current file position. If no items are given then nothing is written to or read from the file. This can be used to position a file for use by other commands ie.g. INPUT for formatted input).

12.1 Byte Input/Output (I/O)

```

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

BGET gets 0 or more bytes from the channel. BPUT puts 0 or more bytes into the channel. For BGET, each item must be a 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 on the file open #3.

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 be 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?

12.2 Unformatted Input/Output (I/O)

It is possible to put or get values in their internal form. The PRINT and INPUT commands of SuperBasic handle formatted I/O, 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 of 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 position, items get internal format data from a file

PUT #channel position, items put internal format data onto a file

GET gets data in internal format from the channel. PUT puts data in internal format into the channel. For GET, each item must be an 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 channel 3, to the 54th byte, and put 2 bytes (integer 42), 6 bytes (floating point 78000), and the 5 characters 'Smith'. Fpoint will be set to 67 (54 + 2 + 6 + 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;

.... B" will force string type;

..... I10 will force integer type.

xyz\$ = 'ab258.z'

...

PUT #3 \ 37,xyz\$(3 to 5) I10

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).

12.3 Truncate File

TRUNCATE #channel position truncate file
If the position is not given, the file will be truncated to the current position.

TRUNCATE #dbchan truncate the file open on channel dbchan

12.4 Flush Buffers

FLUSH #channel flush file buffers
QDOS directory device drivers maintain as much of a file in RAM as possible. A power failure or other accident could result in a file being left in an incomplete state. The FLUSH procedure will ensure that a file is updated without closing it. Closing a file will always cause the file to be flushed. Toolkit II includes an upgrade to the microdrive routines to perform a complete flush. FLUSH will not work with Micro Peripherals disc systems, unless it has been upgraded to version QFLP.

12.5 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) find file position

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 the commands BGET,BPUT, GET or PUT 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 repaired 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 prefetch of parts of a file where the device driver is capable of prefetching.

13 Format Conversions

Toolkit II provides a number of facilities for fixed format I/O. These include binary and hexadecimal conversions as well as fixed format decimal. Most of these are in the form of functions but one new command is included.

13.1 PRINT_USING

PRINT_USING is a fixed format version of the PRINT command:

PRINT_USING #channel, format, list of items to print

The 'format' is a string or string expression containing a template or 'image' of the required output. Within the format string the characters + - # * , . \ ' " ' \$ and @ all have a special meaning. When called, the procedure scans the format string writing out the characters of the string until a special character is found.

If the @ character is found, then the next character is written out, even if it is a special character.

If the character is a " or ', (single or double quotes) then all characters are written out until the next " or '. If the \ is found, then a new line is written out.

All the other special characters appear in 'fields'. For each field an item is taken from the list, and formatted according to the form of the field and written out.

The field determines not only the format of the item, but also the width of the item (equal to the width of the field). The field widths in the examples below are arbitrary.

field	format
#####	if item is string, write string left justified or truncated, otherwise write integer right justified.
*****	write integer right justified empty part of field filled with * (e.g. ***12)
#####.##	fixed point decimal (e.g. 12.67)
*****.**	fixed point decimal, * filled (e.g. **12.67)

##,###.## fixed point decimal, thousands separated by commas (e.g. 1,234.56 or *1,234.56)
-.#####E+08 exponent form (e.g. 2.9979E+08)
+#####E+08 exponent form always includes sign
 The exponent field must start with a sign, one #, and a decimal point (comma or full stop). It must end with four (!!!!!)'s.

Any decimal field may be prefixed or postfixed with a + or -, or enclosed in parentheses. If a field is enclosed in parentheses, then negative values will be written out enclosed in parentheses. If a - sign is used then the sign is only written out if the value is negative; if a + is used, then the sign is always written out. If the sign is at the end of a field, then the sign will follow the value.

Numbers can be written out with either a comma or a full stop as the decimal point. If the field includes only one comma or full stop, then that is the character used as the decimal point. If there is more than one in the field, the last decimal point found (comma or full stop) will be used as the thousands separator, the other used as a decimal point. Long live European unity!

If the decimal point comes at the end of the field, then it will not be printed. This allows currencies to be printed with the thousands separated, but with no decimal point (e.g. 1,234)

Floating currency symbols are inserted into fields using \$ character. The currency symbols are inserted between the \$ and the first # in the field (e.g. \$DM###.## or \$\$###.##). When the value is converted, the currency symbols are 'floated' to the right to meet the value.

For example
 fmt\$ = '@ Charge ***** : (\$Kr###.###,##)
 : ##,###.##+\'

```
PRINT_USING fmt$, 123.45, 123.45, 123.45
PRINT_USING fmt$, -12345.67, -12345.67,
-12345.66
PRINT_USING ' - ##,###!!!!\' , 1234567
```

will print
 \$ Charges ****123.45 : SKr123.45 : 123.45 +
 \$ Charges * -12345.67 : (SKr12.345,67) : 12,345.67 -
 1. 2 3 5 E + 0 6

13.2 Decimal Conversions

These routines convert a value into a decimal number in a string. The number of decimal places represented is fixed, and the exponent form of floating point number is not used.

FDEC\$ (value, field, ndp) fixed format decimal
IDEC\$ (value, field, ndp) scaled fixed format decimal

The 'field' is length of the string returned, 'ndp' is the number of decimal places.
 The three routines are very similar. **FDEC\$** converts the value as it is, whereas **IDEC\$** assumes that the value given is an integral representation in units of the least significant digit displayed. **CDEC\$** is the currency conversion which is similar to **IDEC\$**, except that there are commas every 3 digits.

```
FDEC$ (1234.56,9,2) returns '1234.56'
IDEC$ (123456,9,2) returns '1234.56'
CDEC$ (123456,9,2) returns '1,234.56'
```

If the number of characters is not large enough to hold the value, the string is filled with '*'. The value should be between -2*31 and 2*31 (-2,000,000,000 to +2,000,000,000) for **IDEC\$** and **CDEC\$**, whereas for **FDEC\$** the value multiplied by 10^{-ndp} should be in this range.

13.3 Exponent Conversion

There is one function to convert a value to a string representing the value in exponent form.

FEXP\$ (value, field, ndp) fixed exponent format
 The form has an optional sign and one digit before the decimal point, and 'ndp' digits after the decimal point. The exponent is in the form of 'E' followed by a sign followed by 2 digits. The field must be at least 7 greater than ndp. E.g.

```
FEXP$ (1234.56,12,4) returns '1.2346E+03'
```

13.4 Binary and Hexadecimal

HEX\$ (value, number of bits) convert to hexadecimal
BIN\$ (value, number of bits) convert to binary

These return a string of sufficient length to represent the value of the specified number of bits and the least significant end of the value. In the case of **HEX\$** the number of bits is rounded up to the nearest multiple of 4.

HEX (hexadecimal string) hexadecimal to value
BIN (binary string) binary to value

These convert the string supplied to a value. For **BIN**, any character in the string, whose ASCII value is even, is treated as 0, while any character, whose ASCII value is odd, is treated as 1. E.g. **BIN ('.##.##)** returns the value 5. For **HEX** the 'digits' '0' to '9', 'A' to 'F' and 'a' to 'f' have their conventional meanings. **HEX** will return an error if it encounters a non-recognised character.

14 Display Control

There are three separate facilities provided to extend the display control operations of the QL. They are cursor control, character font control and window reset.

14.1 Cursor Control

The functions **INKEY\$** is designed so that keystrokes may be read from the keyboard without enabling the cursor. Two procedures are supplied to enable and disable the cursor. When the cursor is enabled, it will usually appear solid (inactive). The cursor will start to

flash (active) when the keyboard queue has been switched to the window with the cursor (e.g. by an **INKEY\$**).

CURSEN #channel enable the cursor
CURDIS #channel disable the cursor

Note that while **CURSEN** and **CURDIS** default to channel #1, like most IN/OUT commands, **INKEY\$** defaults to channel #0.

For example:
CURSEN: in\$ = INKEY\$ (#1,250); CURDIS
 will enable the cursor in window #, and wait for up to 5

seconds for a character from the keyboard. If nothing is typed within 5 seconds, then `in$` will be set to a null string ("").

14.2 Character Fount Control

The QL display driver has two character founts built in. The first provides patterns for the value 32 (space) to 127 (copyright), while the second provides patterns for the values 127 (undefined) to 191 (down arrow). For each character the display driver will use the appropriate pattern from the first fount, if there is one, failing that, it will use the appropriate pattern from the second fount, failing that, it will use the first defined pattern in the second fount.

Substitute founts need not have the same range of values as the built in founts. A fount could, for example, be defined to have all values from 128 to 255.

The format of a QL fount is:

```
byte          lowest character value in the fount
byte          number of valid characters - 1
9 bytes of pixels for the lowest character value
9 bytes of pixels for the next character value, etc.
```

The pixels are stored with the top line in the lowest address byte. For each pixel a bit set to one indicates **INK**, a bit set to zero indicates **PAPER**. The leftmost pixel is in bit 6 of the byte.

The character 'g' is stored as:

```
%00000000
%00000000
%00111000
%01000100
%01000100
%01000100
%00111100
%00000100
%00111000
```

The command `CHAR_USE` is used to set or reset one or both character founts

`CHAR_USE #channel, addr1, addr2` `addr1` and `addr2` both point to substitute founts

`CHAR_USE #channel, 0, addr2` the built in first fount will be used, `addr2` points to a substitute second fount

`CHAR_USE 0,0` reset both founts for window #1

15 Memory Management

As QDOS is a multitasking operating system, there may be several jobs running in a QL, and so the amount of freememory may vary unpredictably. No Job may assume that the amount of free memory is fixed. The function `FREE_MEM` may be used to guess at the free memory (defined as the space available for filing system slave blocks less the space required for two (c.f. QL Toolkit: one only) slave blocks.

Temporary space may be allocated in the 'common heap'. This is done with the function `ALCHP` which returns the base address of the space allocated. Individual allocations may be returned to QDOS with the command `RECHP`, or all space allocated is released by the commands `CLCHP` (clear common heap), `CLEAR` or `NEW`.

Functions

`FREE_MEM` find the amount of free memory
`ALCHP (number of bytes)` allocates space in common heap (returns the base address of the space)

The QL display driver assumes that all characters are 5 pixels wide by 9 pixels high. Other sizes are obtained by doubling the pixels or by adding blank pixels between characters. It is possible, with Toolkit II, to set any horizontal and vertical spacing. If the increment is set to less than the current character size (set by `CSIZE`) then extreme caution is required as it will be possible for the display driver to write characters (at the right hand side or bottom of the window) partly outside the window. The windows should not come closer to the bottom or right hand edges of the screen than the amount by which the increment specified is smaller than the character spacing set by `CSIZE`.

`CHAR_INC #channel, x inc, y inc` set the character x and y increments

The channel is defaulted to #1.

The character increments specified are cancelled by a `CSIZE` command.

For example, if there is a 3x6 character fount in a file called 'f3x6' (length 875 bytes), then a 127 column by 36 row screen can be set up:

```
MODE 4
WINDOW 512-2,256-3,0,0 :REMark clear of edges
                           of screen
CSIZE 0,0 :REMark spacing 6x10
CHAR_INC 4,7 :REMark spacing 4x7
fount = ALCHP (875) :REMark reserve space for fount
LBYTES f3x6, fount :REMark load fount
CHAR_USE fount,0 :REMark single fount only
```

14.3 Resetting the Windows

There are two commands for resetting the windows to the turn-on state:

`WMON mode` reset to 'Monitor'
`WTV mode` reset to 'TV' windows

The mode should be 0,4 or 512 for the 4 colour (512 pixel) mode, or 8 or 256 for the 8 colour (256 pixel) mode. Only the window sizes, positions and borders are reset by these commands, the paper strip and ink colours remain unchanged.

Commands

`RECHP base address` return space to common heap
`CLCHP` clear out all allocations in the common heap.

Making large allocations in the common heap and then accessing a drive for the first time, can cause a terrible heap disease called 'large scale fragmentation' where the drive definition blocks become widely scattered in the heap leaving large holes that cease to be available except as heap entries (i.e. you cannot load programs into them). A simple but dangerous cure is to delete the drive definition blocks.

`DEL_DEFB` delete file from definition blocks from common heap

Although there are precautions within the procedure `DEL_DEFB` to minimize damage, care should be taken to avoid using this command while any directory device is active.

16 Procedure Parameters

In QL SuperBasic procedure parameters are handled by substitution: on calling a procedure (or function), the dummy parameters in the procedure definition become the actual parameters in the procedure call. The type and usage of procedure parameters may be found with two functions:

PARTYP (name)	find type of parameter
PARUSE (name)	find usage of parameter
the type is	the usage is
0 null	0 unset
1 string	1 variable
2 floating	2 array
3 integer	

One of the 'tricks' used by many machine code procedures is to use the 'name' of an actual parameter rather than the 'value' (e.g. 'LOAD fred' to load a file name fred). Given the name of a dummy parameter of a procedure, it would be possible to find the name of an actual parameter of a SuperBasic procedure call, but it would be very slow. It is much easier to find the name of an actual parameter, if the position in the parameter list is known.

PARNAM# (parameter number) find name of parameter

For example the program fragment
 pname fred, joe, 'mary'
DEF PROC pname (n1,n2,n3)
PRINT PARNAM\$(1), PARNAM\$(2), PARNAM\$(3)
END DEF pname
 would print 'fred joe ' (the expression has no name).
 One further 'trick' is to use the value of the actual argument if it is a string, otherwise use the name. This is possible in SuperBasic procedures using the slightly untidy **PARSTR\$** function.
PARTSTR\$ (name, parameter number) if parameter 'name' is a string, find the value, else find the name.

For example the program fragment
 pstring fred, joe, 'mary'

DEF PROC pstring (n1,n2,n3)
PRINT PARTSTR\$(n1,1), PARTSTR\$(n2,2),
 PARTSTR\$(n3,3) **END DEF** pstring
 would print 'fred joe mary'.

17 Error Handling

The JS and MG QL ROMs contain unfinished code for error trapping in SuperBasic: Toolkit II corrects some of the remaining problems.

Error handling is invoked by a **WHEN ERROR** clause. Unlike procedure and function definitions, these clauses are static. The error handling within a **WHEN ERROR** clause is set up when the clause is executed, but is only actioned **WHEN** an **ERROR** occurs. This means that a program may have more than one **WHEN ERROR** clause. As each one is executed, the error processing within that clause replaces the previously defined error processing.

The clause is opened with a **WHEN ERROR** statement, and closed with an **END WHEN** statement. Within the clause there may be any normal type of statement. (Although it might be better to avoid calling SuperBasic functions or procedures!) A **WHEN ERROR** clause is exited by a **STOP**, **CONTINUE**, **RETRY**, **RUN**, **LOAD** or **LRUN** command (if you are using Toolkit II). Furthermore the Toolkit II versions of **RUN**, **NEW**, **CLEAR**, **LOAD**, **LRUN**, **MERGE** and **MRUN** reset the error processing (an unfortunate omission from the QL ROMs).

There are some additional facilities intended for use within **WHEN ERROR** clauses.

ERROR FUNCTIONS

These functions correspond to each of the system error codes

{**ERR_NC**, **ERR_NJ**, **ERR_OM**, **ERR_OR**,
ERR_BO, **ERR_NO**, **ERR_NF**, **ERR_EX**, **ERR_IU**,
ERR_EF, **ERR_DF**, **ERR_BN**, **ERR_TE**, **ERR_FF**,
ERR_BP, **ERR_FE**, **ERR_XP**, **ERR_OV**, **ERR_NI**,
ERR_RO, **ERR_BL**}

and return the value **TRUE** if the error, which caused the **WHEN ERROR** clause to be invoked, is of that type. **Do NOT** use **ERR_DF** without Toolkit II.

ERROR information
ERLIN returns the line number where the error occurred
ERNUM returns the error number
ERROR reporting
REPORT #channel reports the last error
REPORT reports the last error to channel #0
REPORT #channel, error number reports the error number given

RETRY and CONTINUE

As the **RETRY** and **CONTINUE** exit from an error clause without resetting the **WHEN ERROR**, they would be useful if they could also be used to exit to a different part of the program. In Toolkit II, **RETRY** and **CONTINUE** can have a line number.

CONTINUE line number continue or retry from a specified line
RETRY line number
100 WHEN ERROR
110 IF ERLIN=200: PRINT #0\oops'; RETRY
120 REPORT
130 STOP
140 END WHEN
150 :
160 do_in x
170 STOP
180 DEFine PROCEDURE do_in (j)
190 FOR i=1 TO 10
200 INPUT #0,'input';j
210 PRINT #0,'value';j
220 END FOR i
230 END DEFine do_in

18 Timekeeping

18.1 Resident Digital Clock

CLOCK default clock in it's own window
CLOCK #channel default clock, 2 rows of 10 chars.
CLOCK #channel, string user defined clock

CLOCK is a procedure to set up a resident digital clock. If no window is specified, then a default window is set in the top RHS 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: any character may be written except \$ or %. If a dollar sign is found in the string then the next character is checked and

\$d or **\$D** will insert the three characters of the day of week,

\$m or **\$M** will insert the three characters of the month.

If a percentage sign is found then

%y or **%Y** will insert the two digit year
%d or **%D** will insert the two digit day of 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.

Example:

MODE 8

OPEN #6,'scr_156x10a32x16'

INK #6,0; PAPER #6,6

CLOCK #6,'QL time %h:%m'

18.2 Alarm Clock

ALARM time set alarm clock to sound at given time

The time should be specified as two numbers: hours (24 hour clock) and minutes:

ALARM 14,30 alarm will sound at half past two

19 Extras

EXTRAS #channel lists the extra facilities linked into SuperBasic
EXTRAS lists the extras to #1

If the output channel is a window, the screen is frozen (CTRL F5) when the window is full. With Toolkit II installed, there are hundreds of extras.

TK2_EXT enforces the Toolkit II definitions of common commands and functions. If, for any reason, some of the Toolkit II extensions have been re-defined, **TK2_EXT** (c.f. **FLP_EXT** floppy disc extensions, **EXP_EXT** expansion unit extensions) will reassert the Toolkit II definitions.

20 Console Driver

20.1 Keyboard Extensions

There are two keyboard extensions to the QL keyboard handling. The first provides a last line recall facility, and the second assigns a string of characters to an 'ALT' keystroke.

<ALT>.<ENTER> keystroke recovers the last line typed

This keystroke recovers (on a per-window basis) the last line typed, provided only that the keyboard buffer is long enough to hold it.

The **ALTKEY** command assigns a string to an 'ALT' keystroke (hold the ALT key down and press another

key). The string itself may contain newline characters, or, if more than one string is given, then there will be an implicit newline between the strings. To add a newline to the end of the string put a null string (' ' or ' ') at the end of the line.

ALTKEY character, strings assign a string to <ALT> character keystroke

For example after the command

ALTKEY 'r','RJOB''SPL''''

when ALT r is pressed, the command 'RJOB''SPL'' will be executed.

ALTKEY 'r' will cancel the **ALTKEY** string for 'r', while

ALTKEY will cancel all **ALTKEY** strings

21 Micro Driver

21.1 Microdrive extensions

There are three extensions to the microdrive filing system. These are available as operating system entry points, but may also be supported as calls from SuperBasic.

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.

21.2 Microdrive Improvements

The **FS.FLUSH** filing system call has been extended to perform a complete flush including header information. This operation may be accessed through the **FLUSH** command.

22 Network Driver

Attempts have been made in Toolkit II to elevate the rather elementary network facilities of the QL to a useful level.

The network performance is dominated by the exceptionally low capability of the network hardware. (If your QL has a pre-D14 serial number then it is highly possible that your network hardware does not work at all, although recent experience has shown that many more pre-D14 QLs have a working network port than generally supposed).

22.1 Network Improvements

Each QL connected to a network should have a unique 'station number' in the range 1 to 63. This is set using the **NET** command.

NET station number

Toolkit II provides a new protocol for broadcast which includes new provisions for handshaking. A broadcast is a message sent from one QL to all other QLs listening to the network. The Toolkit II broadcast protocol has a positive NACK (not acknowledged) handshake, as well as provision for detecting BREAK.

The device names for the network follow the following convention:

NETO_station number	output to station number
NETO_0	send broadcast
NETI_station number	input from station number
NETI_my station number	input from any station
NETI_0	receive a broadcast
NETI_0_buffer size	receive a broadcast into a specified buffer size

When opening a channel to receive a broadcast, a buffer is opened to allow the entire transmission to be received uninterrupted. If no buffer size is specified, then all but 2k bytes of the free memory will be taken. The buffer size should be specified in Kbytes. For example:

NETI_0_10 receive broadcast into a 10 Kbytes buffer.

When a network output channel is closed, then (as with the QL network driver) the network driver will keep trying to send the last buffer for approximately 20 seconds in case the receiving station is busy with its Microdrives. With Toolkit II, however, after about 5 seconds the driver will start checking for a BREAK.

22.2 File Servers

The file server provided in Toolkit II is a program which allows 10 resources attached to one QL to be accessed from another QL. This means that, for example, disc drives attached to just one QL can be accessed from several different QLs. The file server only needs to be running on the QL with the shared 10 resource. This version of the file server is more general than the first version in that the 10 resources may be pure serial devices (such as modems or printers) or windows on the QL display as well as file system devices (such as disc drives).

FSERVE invokes the 'file server'. There may be more than one QL on a network with a file server running: the station number for these QLs should be as low as possible, and should not be greater than 8.

It is possible that files opened across the network may be left open. This can occur if a remote QL is removed from the network, if turned off or is reset. To correct this condition, wait until all other remote QLs have finished their operations on this QL., then remove the file

server and restart with the commands.

RJOB SERVER
FSERVE

22.3 Accessing the File Server

The network files are accessed from remote QLs using a compound device name:

Nstation number__IO device the name of a remote IO device (e.g. **N2__FLP1__** is floppy 1 on the network station 2)

For example

LOAD n2__flp1__fred loads file 'fred' from floppy 1 on network station 2

OPEN__IN #3,n1__flp2__myfile opens 'myfile' on floppy 2 on network station 1

OPEN #3,n1__con__120x20ax0x0 opens a 20 column 2 row window on net station 2

The use of directory default names makes this rather simpler. For example:

PROG__USE__win1__progs by default all programs will be loaded from directory 'progs' on Winchester disk 1 on network station 1

SPL__USE n1__ser set the default spooler destination to SER1 on network station 1

It is possible to hide the network from applications by setting a special name for network file server.

NFS__USE name, network names sets the network file

The 'network names' should be complete directory names, and up to eight network names may be given in the command. Each one of these network names is associated with one of the eight possible directory devices ('name' 1 to 'name' 8).

For example

NFS__USE mdv,n2__flp1__n2__flp2__ sets the network file server name so that any reference to 'mdv1' on this remote QL, will be taken to be reference flp1 on net station 2, likewise 'mdv2' will be taken to be flp2 on net station 2

OPEN__NEW #3,mdv2__fred now this will open file 'fred' on floppy 2 on network station 2

The network names will normally just be a network number followed by a device name as above and will end with an underscore to indicate that the name is a directory. Indeed if the network file server name is to be used with the wild card file maintenance commands, this is the only acceptable form. **QUILL**, however, tends to open a file with the name **DEF__TMP** on **mdv2__**. Clearly, there will be problems if more than one copy of **QUILL** is run across the network at any one time. This can be avoided if the network name for **mdv2__** is set to be a directory:

NFS__USE mdv,n1__flp1__n1__flp2__fred__DEF__TMP opened on **mdv2__** will now appear in directory 'fred' on flp2 on network station 1

22.4 Messaging

The Toolkit II network facilities may also be used for messaging. A window may be opened, a message sent, and a reply read using a simple SuperBasic program. If particularly pretty messages are required, then the graphics facilities of SuperBasic may also be used. The only standard 10 facilities not available across the network are **SD.EXTOP** (extended operations) and **SD.FOUNT** (setting the founts).

For example

```
ch = FOPEN (n2__con__150x10ax0): CLS #ch
INPUT #ch, 'Do you want coffee?';rep$
IF 'y' INSTR rep$ = 1 : PRINT 'Fred wants coffee'
CLS #ch: CLOSE #ch
```

23 Writing programs to use with EX

Programs invoked by EX (or EW or ET) fall into three classifications:

non standard	program header is not standard format;
special	program header is standard but there is an additional flag,
standard	program header is standard.

So far as EX is concerned, the distinction is that a special program must contain the code to open its own Input/Output channels.

At the start of execution a standard or non-standard program will have the following information on the stack:

word	the total number of channels open for this job
[long	the channel ID of the input pipe, if present]
{ long	the channel ID of each filename given in prog spec }

[long	the channel ID of the output pipe, if present]
word	the length of the option string or 0
[bytes	the bytes of the option string]

If there is just one channel open for a Job, then it is opened for read/write unless it is a pipe in which case the direction is implied in the command.

If there is more than one channel open for a Job, then the first channel is the primary input (opened for read only), and the others are opened OVERWRITE. The last channel is the primary output.

A Job should not close the channels supplied, but, when complete, it should commit suicide. Each Job is owned by the next one in the chain, so that when the last job has completed, the entire chain is removed. Committing suicide in this way will put an end of file in the output. Thus an end file from the primary input should, directly or otherwise, indicate to a program that the data is complete.

Special Programs

Standard and special programs have the value \$4AFB in bytes 6 and 7. This is followed by a standard string (length in a word followed by the bytes of the program identification). In the case of a special program heading a further value of \$4AFB (aligned on a word boundary) follows the identification. When the program has been loaded, the option string put on the jobs stack and the input pipe (if it is required) opened and its ID put on the job's stack, then EX will make a call to the address after the second identifying word. Note that the code called will form part of a Basic procedure, not part of an executable program.

On entry to this code, the following registers will be set:

D4.L	0 or 1 if there is an input pipe; ID is not on stack
D5.L	0 or 1 if there is an output pipe; ID is on stack
D6.L	Job ID for this program
D7.L	total number of pipes + file names in prog_spec
A0	address of support routines
A1	pointer to command string
A3,A6	*pointer to first file name (name table)
A4	pointer to job's stack
A5,A6	*pointer beyond last file name (name table)

*these are the standard Basic procedure parameters passing registers.

The file setup procedure should decode the

file_names, open the files required and put the IDs on the stack (A4). Register D0 should be set to the error code on return. D5 must be incremented by the number of channel IDs put on the job's stack. A4 must be maintained as the job's stack pointer. Registers D1 to D7, A0 to A3 and A5 may be treated as volatile.

The routine (A0) to get a file name should be called with the pointer to the appropriate name table entry in A3. D0 is returned as the error code, D1 to D3 are smashed. If D0 is 0, A1 is returned as the pointer to the name (relative to A6). If D0 is positive, A0 is returned as the channel ID of the SuperBasic channel (if the parameter was #n), all other address registers are preserved.

The routine 2(A0) to open a channel should be called with the pointer to the file name in A1 (relative to A6). The file name should not be in the Basic buffer; D3 should hold the access code (overwrite is supported) and the job ID (as passed to the initialisation routine) should be in D6. The error code is returned in D0, while D1 and D2 are smashed, and A1 is returned pointing to the file name used (it may have a default directory in front). If the open fails, A1 will point to the default + given filename. The channel ID is returned in A0 and all other registers are preserved.

In both cases the status register is returned set according to the value of D0.

Appendix A

Appendix and List of Differences

This index lists the SuperBaic extensions in alphabetical order together with the usage (procedure, function, program), the section number describing the facility in detail, the origin of the facility (whether the facility first appeared in the QL ROMs or in the Sinclair QL Toolkit) and principal differences between the facility in the Toolkit II and earlier versions.

This list only includes the most important differences, in many cases there are other improvements over earlier versions.

Name	Usage	Section	Origin	Differences
AJOB	procedure	9	QL Toolkit	accepts Job name
ALARM	program	18	QL Toolkit	resident program
	function	15	QL Toolkit	
ALTKEY	procedure	20	new	
BGET	procedure	12	QL Toolkit	
BIN	function	13	QL Toolkit	
BIN\$	function	13	QL Toolkit	
BPUT	procedure	12	QL Toolkit	
CALL	procedure	7	bug fix	
CDEC\$	function	13	QL Toolkit	
CHAR_USE	procedure	14	QL Toolkit	
CHAR_INC	procedure	- 5	QL Toolkit	
CLCHP	procedure	15	QL Toolkit	
CLEAR	procedure	6	QL	clears WHEN ERROR
CLOCK	program	18	QL Toolkit	configurable program
CLOSE	procedure	10	QL	close multiple files
CONTINUE	procedure	17	QL	specified line number
COPY	procedure	5	QL	uses default directory uses default destination
COPY__O	procedure	5	new	overwrites file
COPY__N	procedure	5	QL	uses default directory uses default destination
COPY__H	procedure	5	new	
CURSEN	procedure	14	QL Toolkit	
CURDIS	procedure	14	QL Toolkit	
DATA__USE	procedure	4	QL Toolkit	
DATAD\$	function	4	new	
DDOWN	procedure	4	new	
DEL__DEFB	procedure	15	new	
DELETE	procedure	5	QL	uses default directory
DEST__USE	procedure	4	new	
DESTD\$	function	4	new	
DIR	procedure	5	QL	uses default directory
DLIST	procedure	4	new	
DO	procedure	6	new	
DNEXT	procedure	4	new	
DUP	procedure	4	new	
ED	procedure	3	QL Toolkit	completely respecified
ERR__DF	function	17	bug fix	
ET	procedure	8	QL Toolkit	
EX	procedure	8	QL Toolkit	
EXEC	procedure	8	QL	now the same as EX
EXEC__W	procedure	8	QL	now the same as EW
EXTRAS	procedure	19	QL Toolkit	
EW	procedure	8	QL Toolkit	
FDAT	function	11	QL Toolkit	
FDEC\$	function	13	QL Toolkit	
FEXP\$	function	13	new	
FLEN	function	11	QL Toolkit	
FLUSH	procedure	12	new	
FNAME\$	function	11	new	

Name	Usage	Section	Origin	Difference
FOP__DIR	function	10	QL Toolkit	finds vacant channel
FOP__IN	function	0	QL Toolkit	finds vacant channel
FOP__NEW	function	10	QL Toolkit	finds vacant channel
FOP__OVER	function	10	QL Toolkit	finds vacant channel
FOPEN	function	10	QL Toolkit	finds vacant channel
FPOS	function	12	QL Toolkit	
FREE__MEM	function	15	QL Toolkit	gives 512 bytes less
FSERVE	program	22	new	
FTST	function	10	new	
FTYP	function	11	QL Toolkit	
FUPDT	function	11	new	
FXTRA	function	11	new	
GET	procedure	12	QL Toolkit	
HEX	function	13	QL Toolkit	
HEX\$	function	13	QL Toolkit	
IDEC\$	function	13	QL Toolkit	
JOBS	function	9	QL Toolkit	
JOBS\$	procedure	9	QL Toolkit	
LBYTES	procedure	7	QL	uses default directory
LOAD	procedure	6	QL	uses default directory clears WHEN ERROR
LRESPR	procedure	7	new	
LRUN	procedure	6	QL	uses default directory clears WHEN ERROR
MERGE	procedure	6	QL	uses default directory clears WHEN ERROR
MRUN	procedure	6	QL	uses default directory clears WHEN ERROR
NEW	procedure	6	QL	clears WHEN ERROR
NFS__USE	function	22	new	
NXJOB	function	9	QL Toolkit	
OJOB	function	9	QL Toolkit	
OPEN	procedure	10	QL Toolkit	uses default directory
OPEN__DIR	procedure	10new	new	uses default directory
OPEN__IN	procedure	10	QL	uses default directory
OPEN__NEW	procedure	10	QL	uses default directory
OPEN__OVER	procedure	10new	new	uses default directory
PARNAM\$	function	16	new	
PARSTR\$	function	16	new	
PARTYP	function	16	QL Toolkit	
PARUSE	function	16	QL Toolkit	
PJOB	function	9	QL Toolkit	
PRINT__USING	procedure	13	new	
PROG__USE	procedure	3	QL Toolkit	
PROGD\$	function	3	new	
PUT	procedure	12	QL Toolkit	
RECHP	procedure	15	QL Toolkit	
RENAME	procedure	5	QL Toolkit	
RETRY	procedure	17	QL	specified line number
RJOB	procedure	9	QL Toolkit	accepts Job name
RUN	procedure	6	QL	clears WHEN ERROR
SAVE	procedure	6	QL	uses default directory
SAVE__O	procedure	6	new	overwrites file
SBYTES	procedure	7	QL	uses default directory
SBYTES__O	procedure	7	new	overwrites file
SEXEC	procedure	7	QL	uses default directory
SEXEC__O	procedure	7	new	overwrites file
SPJOB	procedure	9	QL Toolkit	accepts Job name
SPL	program	5	QL Toolkit	simplified destination
SPL__USE	procedure	4	QL Toolkit	
SPLF	program	5	new	adds form feed to file
STAT	procedure	5	QL Toolkit	
STOP	procedure	6	QL	clears WHEN ERROR
TK2__EXT	procedure	20	new	
TRUNCATE	procedure	12	QL Toolkit	position may be specified
VIEW	procedure	3	QL Toolkit	

Name	Usage	Section Origin	Difference	Name	Usage	Section Origin	Difference
WCOPY	procedure	5	new	WMON	procedure	14	QL Toolkit
			defaults to command window uses default destination	WREN	procedure	5	new
			defaults to command window and uses default destination	WTV	procedure	14	QL Toolkit
WDEL window	procedure	5	QL Toolkit	WSTAT	procedure	5	QL Toolkit
WDIR	procedure	5	QL Toolkit				

Appendix B

The appendix illustrates the use of Toolkit II facilities with the GST assembler and linker. (The version used by QJUMP is supplied by GST with their QC compiler; QC is well worth buying just to get the assembler and linker!)

The programs accept a wide variety of options on their command line. This command line can be passed to the programs in the parameter string of the EX command. Unfortunately the programs do not attempt to find the default data directory, so it is necessary to add this to the file names in the command line.

The assembler is called ASM and the linker LINK. Filenames can be passed to these procedures as strings or names.

100 REMark assemble a relocatable file

110:

120 DEFine PROCedure asr (file\$)

130 EX asm: DATAD\$ & PARSTR\$ (file\$,1) & ' -list & "errors scr"

140 END DEFine asr

150:

160 REMark assemble with listing

170:

180 DEFine PROCedure asl (file\$)

190 EW asm: DATAD\$ & PARSTR\$ (file\$,1) & ' -list ser -nosym'

200 END DEFine asl

210:

220 REMark link program

230:

240 DEFine PROCedure lk (file\$)

250 EX link: DATAD\$ & PARSTR\$ (file\$,1) & ' -with ' & DATAD & "link -nolist"

260 END DEFine lk

If the default directory is 'FLP1__JUNK__', then the procedure calls:

ASL 'table' and LK master

will create the command parameter strings to the assembler and linker

'FLP1__JUNK__table -list ser -nosym' and

'FLP1__JUNK__master -with FLP1__JUNK__link -nolist

Appendix C

QL Network Protocols

Standard QL Handshake

The Standard QL handshaking network protocol is compatible with the Sinclair Spectrum protocol. It comprises 11 phases

	sender	receiver
a) scout		
1) gap	waiting for 3ms for activity, if activity occurs: restart	
2) wait		waiting for activity (a scout)
3) scout	send a scout of duration <530us, if contention occurs: restart	wait for 530us
b) header		
4) hactiv	set net active 22us for each byte 11.2us start (inactive) bit, 8*11.2us data bits, 5*11.2us stop (active) bits	wait for active for each byte wait for start (inactive) bit, read 8 data bits, if fails: restart
5) hbytes		
6) Hackw	wait for 2.5ms for active, if not active: restart	set net active 22us

7) hackbt wait for start bit, read 8 data bits, if error: restart send 11.2us start bit 8 data bits 00000001

c) data

8) dactiv set net active 22us for each byte 11.2us start (inactive) bit, 8*11.2us data bits, 5*11.2us stop (active) bits wait for active for each byte wait for start (inactive) bit, read 8 data bits, if fails: restart

10) dackw wait for 2.5ms for active, if not active: restart set net active 22us

11) dackbt wait for start bit, read 8 data bits, if error: restart send 11.2us start bit 8 data bits 00000001

The entire protocol is synchronised by a period of inactivity at least 2.8ms long.

The header is eight bytes long in the following format:

destination station number
 sending station number
 block number (high byte)
 block number (low byte)
 block type (0 normal, 1=last block of file)
 number of bytes in block (0 to 255)

data checksum
header checksum

If the number of bytes in a block is 0, 256 data bytes are actually sent.

The checksums are formed by simple addition; if there are two single bit errors in the most significant bit (the most common type of error) within one block, then the errors will pass undetected.

If the block number received in a header is not equal to the block number required, then the header and data block are acknowledged but ignored.

The protocol is not proof against a failure on the last block transmitted where the receiver has accepted the block, but the sender has missed the acknowledge. In this case the sender will keep re-transmitting the block until it times out (about 20s).

Toolkit II Broadcast

Toolkit II has a special version of this protocol for network broadcast. This has an extended scout to allow time for the receiver to interrogate the IPC without missing the scout, and it has an active acknowledge/not acknowledge. The protocol has been defined in such a way that future network drivers can be more flexible than the Toolkit II driver.

	sender	receiver
a) scout		
1) gap	waiting for 3ms for activity, if no activity occurs: restart	
2) wait		waiting for activity (a scout) every 20ms check IPC for BREAK wait for 530us
3) scout	send a scout of duration x530us, if contention occurs: restart	
4) scext	restart	send a scout extension of 5ms active
b) header		
5) hbytes	for each byte 11.2us for start (inactive) bit, 8*11.2us data bits, 5*11.2us stop (active) bits	for each byte wait for start (inactive) bit, read 8 data bits, if fails: nack
6) hwait	leaving net active, wait 1ms	

c) data

7) dbytes for each byte 11.2us wait (inactive) bit, 8*11.2us data bits, 5*11.2us stop (active) bits

for each byte wait for start (inactive) bit, read 8 data bits if fails: nack

8) dack inactive net and wait 1ms for active: if fails, restart

within 500us set net active and wait 5ms, do any processing required and when ready for next packet, inactivate and restart

d) Not acknowledge

9) nack wait for inactive

wait for 2.8us of active or inactive, if inactive: restart

10) nackw wait 500us for active: timeout is ok, active is fail

wait 200us for active, if active, restart if inactive activate 500us (nack)

A broadcast acknowledge is 5ms active followed by more than 400us inactive. A broadcast not acknowledge is no response or 5ms active followed by 200us to 300us inactive, followed by more than 200us active.

Toolkit II Server Protocol

The Toolkit II server protocol is physically the same as the Standard QL protocol, but the header has been slightly changed to improve the checksum, to allow blocks of up to 1000 bytes to be sent and to distinguish server transactions. A server header cannot be confused with standard header.

wait for 500us for active: timeout is ok, active is fail	wait 200us for active, if active: restart, if inactive activate 500us (nack)
--	--

