MIRACLE SYSTEMS LTD

QDOS - SMSQ Compatible Products

USER MANUAL

20 Mow Barton, Yate, Bristol, BS17 5NF, UK
Telephone/Fax: (01454) 883602

Miracle Systems Ltd

QXL SUPPLEMENT

20 Mow Barton, Yate, Bristol, BS17 5NF, UK
Telephone/Fax: (01454) 883602

QXL INSTALLATION

WARNING: the QXL contains static sensitive devices and should be kept in the black anti-static bag supplied when not plugged into the computer.

Owners of our QL Hard Disk should not load WIN_REXT on the QXL.

INSTALLING THE QXL HARDWARE

If your PC offers a turbo mode this may have to be disabled before installing the QXL. Some PCs also have the ability to switch off wait states on the I/O bus. This facility may have to be disabled in the CMOS setup to allow wait states. Failure to do the above will not cause damage but may result in unreliable operation. The software will only support one QXL so put only one in your computer.

The QXL will work in an 8-bit or 16-bit ISA slot. It comes configured for port address 2B0 hex so should work in most machines without alteration. Leave the switches on the QXL alone unless you are sure that they need changing.

If you are at all unsure as to how to fit an expansion card to your computer please read the computer's User Manual first. Make sure that the power to the computer is switched off before removing its cover. Choose a slot for the QXL; a 16-bit slot consists of two connectors in line, one slightly longer than the other; an 8-bit slot consists of just one connector. Once a slot has been chosen remove the appropriate slot cover and retain the screw if there is one. Push the QXL into the chosen slot by its edge taking care not to apply pressure to any components and if the slot cover was retained by a screw use it to secure the QXL's bracket. Replace the computer's cover then switch it on.

INSTALLING THE QXL SOFTWARE

The software supplied with the QXL can be installed on a hard disk but this is not obligatory.

Starting from hard disk

Assuming the QXL diskette is in drive A:, from the C: prompt type the following lines pressing ENTER after each line.

```
CD \
MKDIR QXL
CD QXL
COPY A:SMSQ.EXE C:
```

Entering the lines above will put a subdirectory called QXL on the hard disk and transfer the contents of the diskette in drive A: into that subdirectory. To activate the QXL, after typing the lines above, simply type : **SMSQ** then press ENTER.

To make the computer power up as a QL it is necessary to edit the file called AUTOEXEC.BAT which should be in the root directory of the hard disk, reached by entering CD\ . The last two lines of AUTOEXEC.BAT need to be

CD QXL
SMSQ

See the computer's manual on how to do this.

Giving the QXL access to the hard disk

Once the QXL is running it is possible to create a hard disk. The hard disk is called WIN1_ and must be formatted before use. The size is specified in Megabytes and follows the device name. For example, a 100 Megabyte hard disk is created with

FORMAT "WIN1_100"

Do not worry. This does not format the PC's hard disk, just creates a file 100 Megabytes long on the C: drive called QXL.WIN in the root directory.

There is currently no way of accessing MSDOS files on the hard disk.

To boot from WIN1_ make sure there is no diskette in FLP1_ before starting the QXL, an attempt is then made to load a file called WIN1_BOOT.

Starting from floppy disk

Reliable as they are floppy disks wear out. This means you should not use the floppy disk supplied by us except when making a working copy of it.

The easiest way to make a working copy is with the DOS command DISKCOPY. If your machine does not have a hard disk and the diskette with the DISKCOPY command on it is in drive A: type

DISKCOPY A: A:

then press ENTER. The computer will prompt for the SOURCE diskette which is the master diskette from MIRACLE SYSTEMS so put this in drive A: and follow the instructions. After a while another prompt will appear asking for a DESTINATION diskette so put the diskette intended for use as the working copy of the QXL software in drive A:. The diskettes may have to be changed again before copying is complete. The diskette used as the DESTINATION does not need formatting first but be careful as any files on it will be lost.

To start the QXL, with the working copy in drive A:, enter **SMSQ**

RETURNING TO THE DOS PROMPT

To return to the DOS prompt hold down the CTRL key and press the key marked ScrollLock then release both keys. Control will be passed back to DOS.

If you wish to restart the QXL from where you left off enter

SMSQ/

To reset the QXL having left it with CTRL-ScrollLock reenter

SMSQ

EXECUTING PROGRAMS ON THE QXL

A list of differences between the QXL's SBASIC and the QL's SuperBASIC is contained in a later section of this manual.

To start a copy of QUILL that has been changed to run from FLP instead of MDV, place the diskette in drive A:, which from now on is referred to as FLP1_, and enter

EXEC_W "FLP1_QUILL"

Replace QUILL with ABACUS, etc for other PSION programs.

THE PARALLEL PORT

The QXL has access to the host PC's parallel port, LPT1:, via a device called PAR. To give software that can only use SER access to the parallel port there is a command called PAR_USE. QUILL can be configured to use PAR as its default printer using INSTALL_BAS. Alternatively, enter

PAR_USE "SER"

before starting QUILL to achieve the same effect. All output to SER, SER1 or SER2 will then be redirected to PAR.

To restore access to the serial ports enter

PAR_USE "PAR"

THE SERIAL PORTS

The QXL has access to the host PC's serial ports, COM1: and COM2:, via device names SER1 and SER2. The QXL software comes configured with SER2 disabled to allow a serial mouse to work from COM2:. The Pointer Environment must be loaded to use the mouse and the PC's mouse driver must have been installed; see the instructions that came with the mouse for its installation.

PLEASE NOTE: if your PC does not have a COM2: or your mouse does not allow operation in COM2: and you wish to use the mouse under the Pointer Environment the software needs to be reconfigured. If your PC has no COM2: set SER2 to none, an AT needs to be set to COM2/IRQ3 and a PC/XT needs to be set to COM2/IRQ4. If the configuration is altered the computer must be rebooted; just restarting the QXL is not enough. See the section POINTER ENVIRONMENT USERS for more information.

Additional facility

There is an additional facility added to the SER and PAR drivers. This is "A" mode and can be used instead of the existing "C" mode and will convert the <LF> character into <CR><LF>.

If the following was entered:

```
OPEN #3,"SER1"
PRINT #3,"This is line one"
PRINT #3,"This is line two"
PRINT #3,"This is line three"
CLOSE #3
```

and the output on the printer was:

```
This is line one
        This is line two
                This is line three
```

then if "SER1A" was used instead of "SER1" the print-out would be correct.

The correct output could also have been obtained by changing the switches on the printer but the "A" option is easier and doesn't alter the operation of the printer with other software.

See the the Concepts section under Devices in the QL User Guide for more information on the "C" mode if necessary.

The way to enable or disable the serial ports is using the QPAC2 "Config" program supplied on the QXL disk.

NETWORK PORTS

The QXL can access devices over the network or act as a file server in the same way as a QL fitted with a SUPER GOLD CARD, GOLD CARD or TRUMP CARD.

FORMATTING DISKETTES

Diskettes must first be formatted under MSDOS. They can then be cross-formatted simply by using the standard FORMAT command from within SMSQ.

SCREEN HANDLING

Mode 8 does not support FLASH.

POINTER ENVIRONMENT USERS

The POINTER ENVIRONMENT files Ptr_Gen, WMan, and Hot_RExt are included on the QXL software diskette in files called PTRGEN, WMAN and HOTREXT respectively. These are not special versions only for the QXL but supersede earlier versions. Don't forget to add the underscore when copying them from the diskette.

e.g. To copy these three files to a subdirectory on "Win1_" called "PtrEnv_" from "Flp1_":

```
COPY "Flp1_PTRGEN" to "Win1_PtrEnv_Ptr_Gen"
COPY "Flp1_WMAN" to "Win1_PtrEnv_WMan"
COPY "Flp1_HOTREXT" to "Win1_PtrEnv_Hot_RExt"
```

The Qjump CONFIG program (also supplied) can be used on the SMSQ file to provide access to higher resolution screens. Pointer Environment and SBASIC can use the higher resolutions; Xchange 3.90I (available from IQLR) also makes use of the higher resolutions.

The supplied diskette is in MSDOS format and so cannot support executable files. To make CONFIG executable get the QXL up and running then put the QXL diskette into drive A: and enter the following:

```
Adr=RESPR(5510)
LBYTES "Flp1_Config",Adr
SEXEC "Win1_Config",Adr,5510,1792
```

which will leave the executable version of Config on Win1_.

To use Config, put an MSDOS format diskette with SMSQ on it into Flp1_ and then enter:

```
EXEC "Win1_Config"
```

then press CTRL-c until the cursor flashes in the Config window and follow the on screen instructions. The name of the file to configure is "Flp1_SMSQ.EXE"

If you are using the QXL from hard disk it is necessary to go to MSDOS, with CTRL-ScrollLock, and copy the file called SMSQ.EXE to an MSDOS format floppy diskette before it can be modified with CONFIG. Re-enter the QXL by entering SMSQ/ and the file can be modified by CONFIG.

KEYBOARDS

CONFIG can also be used to change the keyboard from its default of English to German or French and enable or disable the parallel port and serial ports. Keyboards currently supported are:

Numeric Code	Alpha code	Type
1	USA	American
33	F	French
44	GB	English
49	D	German

How To Configure The QXL For Use At Different Addresses

If you are sure that the I/O port address of 2B0H for which the QXL comes configured will clash in your system consult the table below. In practice an address clash will be fairly obvious; the system refusing to start properly or the QXL not working together with, for example, a network card that no longer functions. If, starting the QXL for the first time, you get the error message:

The QXL at IO address 02B0h is not responding

it is a sure sign that something is wrong. All QXLs are tested before shipping so unless it has been damaged in transit the above message may indicate an address clash or the PC is in turbo mode or the CMOS setup has been set for zero wait states on I/O cycles.

A computer fitted with the usual peripherals will not cause problems because they have predetermined addresses, none of which coincide with the QXL's address. Usual peripherals are:

(1) a display card such as for VGA,
(2) two floppy disks,
(3) a hard disk,
(4) two serial ports,
(5) a parallel port,
(6) a games port.

(2) to (6) may all be on a single "Super IO Card" but the addresses used remain the same.

QXL SWITCH SETTING

The switches on the QXL allow the following addresses :

SWITCH NUMBER 1	2	3	4	IO PORT ADDRESS	FUNCTION ON COMPUTER
OFF	OFF	OFF	ON	280H	Not used
ON	OFF	OFF	ON	290H	Not used
OFF	ON	OFF	ON	2A0H	Not used
ON	ON	OFF	ON	2B0H	QXL default
OFF	OFF	ON	ON	2C0H	Not used
ON	OFF	ON	ON	2D0H	Not used
OFF	ON	ON	ON	2E0H	Not used
ON	ON	ON	ON	None	-
OFF	OFF	OFF	OFF	300H	Prototype card
ON	OFF	OFF	OFF	310H	Prototype card
OFF	ON	OFF	OFF	320H	XT hard disk
ON	ON	OFF	OFF	330H	Not used
OFF	OFF	ON	OFF	340H	Not used
ON	OFF	ON	OFF	350H	Not used
OFF	ON	ON	OFF	360H	Not used
ON	ON	ON	OFF	None	-

If the switches on the QXL are changed the software must be informed. For example if you have decided to place the QXL at address 290H enter the following:

SMSQ 290

The address supplied will be written to a file called QXL.DAT. This saves having to remember the QXL's address. If the QXL has been started from a write protected floppy diskette you will get an error message. Set the write protect tab to write enabled and press R to retry the write. Don't forget to set the tab back to write protected once the QXL is up and running.

C1 3F8
C2 2F8
P 378

1 INTRODUCTION

SMSQ/QXL is based on the SMS Kernel which was designed to provide a QDOS compatible interface. The kernel has been modified to improve compatibility with most of the "DIRTY TRICKS" which QL programmers were either forced to use or used to satisfy their perverted sense of fun.

The Kernal itself (Memory Management, Task Management, Scheduling and IO) has also been extended to provide facilities which were not available with QDOS. It is now an over inflated 10K bytes. Despite this inflation, the SMSQ operating system kernel remains more efficient than the old QDOS kernel.

Superbasic has been replaced by SBASIC which is a threaded code INTERPRETER which executes at speeds more often associated with compiled SuperBASIC than INTERPRETED SuperBASIC. There is no longer any need to compile SuperBASIC programs: You can just execute them.

In addition, SMSQ/QXL is supplied with entirely new filing system device drivers which allow "FOREIGN" disk formats to be recognised and new formats to be added "AT RUN TIME".

2 NEW AND MODIFIED FACILITIES

SMSQ/QXL includes all the QL SuperBASIC commands, the TK2 commands (activated by the TK2_EXT command - this manual does not concern itself with standard SuperBASIC or TK2 commands.) SMSQ/QXL supports 99% of SuperBASIC. SMSQ/QXL supports all the devices which were supported by the drivers supplied with the TRUMP CARD and GOLD CARDS.

There are, however, a number of significant new facilities and improvements, some of which may be familiar to some useres.

FACILITY	USAGE OR DIFFERENCE	SECTION
$nnn %nnn	Hexadecimal and binary values accepted	4.1
ATAN	ATAN (X,Y) yields a four quadrant result	7.10
BAUD	Independent baud rates	10.3
BGET BPUT	Transfer multiple bytes to and from strings	7.9
CACHE_ON CACHE_OFF	Turn internal caches on or off	3.2
DEV	A defaulting filing system device	11.3
DEV_LIST	Lists the current DEVs	11.3.2
DEV_NEXT	Enquires the next DEV for a DEV	11.3.3
DEV_USE	Sets the real device for a DEV	11.3.1

FACILITY	USAGE OR DIFFERENCE	SECTION
DEV_USE$	Enquires the real device for a DEV	11.3.3
DEVTYPE	Find the type of device open as a channel	5
END FOR, END REPEAT	Do not need names	4.5.4
EX EW EXEC EXEC_W	Extended to execute SBASIC programs	8.3
EPROM_LOAD	Loads and initialises a "QL EPROM cartridge"	7.6
EXIT	Does not need a name	4.5.4
IF	Multiple nested inline IFs, nesting is checked	4.2
IO_PRIORITY	Set the priority of IO retry scheduling	3.1
JOB_NAME	Sets the job name for SBASIC jobs	8.2
KBD_TABLE	Uses international codes to set keyboard tables	10.2.3
LANG_USE	Sets the message language	10.2.1
LANGUAGE($)	Language enquiry	10.2.2
LOAD LRUN	Accept QLOAD _SAV files and save filename	7.1
LBYTES	Accepts channel number in place of name	7.5
LRESPR	If used to load extensions within an SBASIC job other than job 0, the extensions are private to that job	8.4
MERGE MRUN	Accept QLiberator _SAV files	7.1
NEXT	Does not need a name	4.5.3
NUL	A bottomless bin for output or endless input	11.1
PEEK etc.	Extended to access system and SBASIC variables	7.8
PEEK$	PEEKs multiple bytes	7.7
PIPE	Named or unnamed pipes for intertask communications	11.2
POKE etc.	Extended to access system and SBASIC variables	7.8
POKE$	POKEs multiple bytes	7.7
PROT_DATE	Protect the real time clock	7.11
QLOAD QLRUN	QLiberator compatible quick load for _SAV file	7.3
QMERGE QMRUN	QLiberator compatible quick merge for _SAV file	7.4
QSAVE QSAVE_O	QLiberator compatible save to _SAV file	7.4
QUIT	Removes this SBASIC job	8.3
REPeat	Does not need a name	4.5.5
SAVE SAVE_O	Use previously defined filename, update version	7.2
SBASIC	Starts an SBASIC daughter	8.1
SBYTES SBYTES_O	Accepts channel number in place of filename	7.5
SELect	Both integer and floating point SELects	4.3
SEXEC SEXEC_O	Accepts channel number in place of name	7.5
SLUG	Slows the machine down	3.3
TRA	Language selectable and language independent	10.2.4
VER$	Minerva compatible	7.12
WMON WTV	Allow the SBASIC windows to be offset	8.1

3 SMSQ PERFORMANCE

In general, SMSQ is more efficient than QDOS. There are, however, a number of policy differences which are either accidental because, unlike other"QDOS compatible" systems SMSQ is not based on QDOS but is completely redesigned, or deliberate because certain QDOS policies have shown to be less than ideal.

In particular, the IO retry scheduling policy is completely different. This results in a very much higher priority for retry operations which greatly improves responsiveness of a heavily loaded system at the cost of a modest reduction in crude performance (typically 10%). If crude performance is important to you, you can reduce the IO priority to QDOS levels.

3.1 IO_PRIORITY: The IO_PRIORITY (priority) command sets the priority of the IO retry operations. In effect, this sets a limit on the time spent by the scheduler retrying IO operations.

A priority of one sets the IO retry scheduling policy to the same as QDOS, thus giving a similar level of response but with a higher crude performance.

IO_PRIORITY 1	QDOS levels of response, higher crude performance
IO_PRIORITY 2	QDOS levels of response, better response under load
IO_PRIORITY 10	Much better response under load, degraded performance
IO_PRIORITY 1000	Maximum response, the performance depends on the number of jobs waiting for input.

3.2 CACHE_ON CACHE_OFF: The performance of the more powerful machines depends on the use of the internal cache memory. For the MC680X0 series processors, the implementation of the caches is less than perfect. As well as introducing unnecessary overheads on operating system calls (slightly improved in the MC68040) the MC680X0 cache policy is incompatible with certain programming techniques. It may, therefore, he necessary to disable the internal caches.

No provision is made for disabling the external caches (where these exist) as none of these external caches seem to suffer from the design flaws of the MC680X0 series.

CACHE_OFF	Turn the caches off to run naughty software
CACHE_ON	And turn back on again

3.3 SLUG: The designers of SMSQ have spent much time and effort trying to make the system fairly efficient. Their efforts seem not to be appreciated. Some people will always complain! SLUG (slug factor) will slug your machine by a well defined factor.

SLUG 2	Half speed ahead
SLUG 5	Dead slow
SLUG 1	Full ahead both

4 SBASIC / SuperBASIC LANGUAGE DIFFERENCES

Some differences between SBASIC and SuperBASIC may be accidental. There are, however, a number of known, deliberate differences. Most of these differences are extentions to SuperBASIC. In some cases, however, limitations have been introduced to reduce the chances of difficult-to-track-down program errors.

4.1 Hexadecimal and Binary Values: Hexadecimal and binary values may be included directly in SBASIC source. Hexadecimal values are preceded by a $. Binary values by a %.

```
IF a% && %1001                    check bits 3 and 0 of a%
IF PEEK_L($28000)=$534D5351       check if SMSQ (very naughty)
```

4.2 IF Clauses: Multiple "in-line" IF clauses can be nested on one line. SBASIC checks for incorrectly nested IF clauses.

4.3 SELect Clauses: SELect clauses may select an action on the value of an integer variable (integer SELect) or on the value of a floating point variable or expression (floating point SELect). Integer SELect is more efficient. SBASIC checks for incorrectly nested or inconsistent SELect clauses.

4.4 WHEN ERRor: WHEN ERRor is suppressed within the command line to stop SBASIC rushing off into your error processing if you mistype a command. You can turn off WHEN ERRor by executing an empty WHEN ERRor clause.

```
100 WHEN ERRor
110     CONTINUE          :REMark ignore errors
120 END WHEN
130 A=1/0                 :REMark no error
140 WHEN ERRor            :REMark restore error processing
150 END WHEN
160 A=1/0                 :REMark  BANG!!
```

4.5 Loop Handling

4.5.1 FOR Loop Types: SuperBASIC requires FOR loops to have a floating point control variable. SBASIC allows both floating point and integer control variables. Integer FOR loops are more efficient than floating point FOR loops: particularly if the control variable is used to index an array.

```
FOR i%=0 TO maxd% : array(i%)=array(i%)*2     is preferred to
FOR i=0 TO maxd : array(i)=array(i)*2         which is less efficient
```

N.B. the type is determined before the program is executed.

4.5.2 In-Line Loops: Whereas SuperBASIC only allows a single structure to be defined "in-line", SBASIC allows many loops (and other structures) to be nested in-line without END statements:

```
100 FOR i=1 TO n : FOR j=1 TO m : a(i,j)=a(i,j)+b(i,j)
```

4.5.3 The "NEXT Bug": The "NEXT bug" reported in many articles about SuperBASIC, which many people have asked to be fixed, has not been fixed. IT IS NOT A BUG. NEXT is defined to fall through to the next statement when the loop is exhausted. It does not go to the statement after the END FOR (which may not be present). If that is what you wish to do, follow the NEXT by an EXIT.

4.5.4 Unnamed NEXT, EXIT and END Statements:

Loop structures are "opened" with a FOR or REPeat statement and closed with an END FOR or END REPeat statement. SuperBASIC requires all loop closing statements as well as the intermediate NEXT and EXIT statements to identify the loop to which they apply. SBASIC, on the other hand, will accept unnamed NEXT, EXIT, END FOR and END REPeat statements. These are applied to the most recent (innermost) unclosed loop structure.

```
100 FOR i=1 TO 10
110     FOR j=1 TO 10
120         IF a(i,j)<0:EXIT          implicitly EXIT j
130         sum=sum+a(i,j)
140     END FOR                       implicitly  END FOR j, closes FOR j
150     IF sum<100:NEXT               loop j is closed, so this is NEXT i
160     PRINT i,sum
170     sum=0
180 END FOR                           implicitly END FOR i, closes FOR i
```

4.5.5 REPeat Loops: Whereas SuperBASIC requires all REPeat clauses to have a name, SBASIC allows unnamed REPeats. These unnamed REPeats may be combined with unnamed NEXT, EXIT and END REPeat statements.

```
100 REPeat
110     a$=INKEY$(-1)
120     IF a$=ESC$:EXIT               goes to 210 (outer loop)
130     IF a$<>'S':NEXT               goes to 110 (outer loop)
140     REPeat
150         a$=INKEY$(-1)
160         IF a$=ESC$:EXIT           goes to 190 (inner loop)
170         x$=x$&a$
180     END REPeat                    goes to 150 (inner loop)
190     IF LEN(x$)>20:EXIT            goes to 210 (outer loop)
200 END REPeat                        goes to 110 (outer loop)
210 PRINT 'DONE'
```

4.6 Multiple Index Lists and String Slicing: For various reasons SBASIC does not support multiple index lists.

```
100 DIM a(10,10,10)
110 a(3,4)(5)=345        OK for SuperBASIC, SBASIC will not handle this
120 a(3,4,5)=345         Means the same, is easier to type and SBASIC likes it
```

To make up for this limitation, SBASIC allows you to slice strings at any point in an expression.

```
200 a$=2468(3)                  Sets a$ to '6' in SBASIC, prohibited in SuperBASIC
210 ax=1234
220 a$=('abcdef'&ax)(5 to 8)    Sets a$ to 'ef12' in SBASIC
230 b$='abcdefghi'
240 a$=b$(2 to 7)(3 to 5)(2)    Sets a$ to 'e' in either SBASIC or SuperBASIC
```

Also, in SBASIC, the default range for a string or element of a string array is always (1 TO LEN(string)) and zero length slices are accepted at both ends of a string. A$(1 to 0) or A$(LEN(string)+1 TO LEN(string)) are both null strings).

5 Writing Compiler Compatible Programs

SuperBASIC compatible programs which are written in such a way as to be used both compiled and interpreted by SuperBASIC often have a small code fragment at the start to allow for the differences in compiled and interpreted environments.

The problem is not that SBASIC is "incompatible" with these code fragments but SBASIC is compatible with SuperBASIC in a way which the two "compiled" SuperBASICs are not. The simplest way to avoid these problems is to give up using compiled BASIC and remove the junk from your programs. If, on the other hand, you wish to continue using compiled BASIC and also wish to use the programs in SBASIC daughter jobs, you may require some code changes.

There are three principal differences between the SuperBASIC environment and the QLiberator and Turbo environments.

1. When executing in compiled form, the program will probably not be requiring windows #0, #1 and #2 in the same form as when it is being interpreted by SuperBASIC. In particular:

- channel #0 (the command channel) may not be required at all in the compiled version, but it is essential to keep it open in the SuperBASIC version otherwise no commands can ever be given again;

- a compiled program may be started with no windows open, a program interpreted by SuperBASIC will (usually) start with windows #0, #1 and #2 open.

This distinction is not so much a difference between compiled and not compiled, but is a difference between interpreting a program within the permanent SuperBASIC interpreter and executing a transient program.

2. An interpreted program may be interrupted and rerun (so that the starting state may be different each time), while a compiled program will always start "clean" (always having the same starting state).

3. An interpreted program will report error messages to window #0 while compiled programs have their own error message facilities.

From the point of view of the last two differences, SBASIC is always much closer to SuperBASIC than to a compiled BASIC. For the first (and most important difference) SBASIC can behave either like a compiled BASIC or SuperBASIC.

- If SBASIC is started off with an SBASIC command, then SBASIC behaves like SuperBASIC: window #0 (at least) is open.

- If SBASIC is started off with an EX (etc.) command or from a HOTKEY or QPAC2 EXEC menu, then SBASIC behaves more like a compiled program: there are no windows open by default and window #0 is not required.

Unfortunately, the code that usually appears at the start of these compatible programs does not distinguish between compiled and interpreted environments, but between job 0 and other jobs.

```
100 IF JOB$(-1)<>""                 :REMark is it a named job (NOT SuperBASIC)
110     CLOSE #0,#2                 :REMark close spare windows in case
120     OPEN #1,"Con_512x256a0x0"   :REMark our #1
130 ELSE
140     WINDOW 512,256,0,0          :REMark for SuperBASIC, just set #1
150 END IF
160 CLS
```

When used in an SBASIC daughter job, this will treat SBASIC as compiled whereas it should possibly be treated as interpreted as SBASIC programs can be re-run.

The problem cannot be resolved by using a function to distinguish between compiled, SuperBASIC and SBASIC, as there is no such function in SuperBASIC and it cannot be assumed that a suitable extension has been loaded.

SBASIC jobs are, however, always called SBASIC until the name is set by the JOB_NAME command.

The best approach would be to have program start up code which is sensitive to the environment and not having a different behaviour just because the job number is 0 or the job has no name. This is, however, not practical with the old QL BASIC compilers.

The least bad solution may be to have a "four way switch" at the start of the program.

```
100 my$='myjob':j$=JOB$(-1)          :REMark set my assumed and real names
110 IF j$=""                         :REMark is it an unname job (SuperBASIC):
120     do SuperBASIC or SBASIC job 0 fiddles
130 END IF
140 IF j$='SBASIC'                   :REMark is it start of an SBASIC daughter?
150     do SBASIC daughter initialisation
160     JOB_NAME my$                 :REMark from now on it is a named job
170     j$=""                        :REMark no further action required
180 END IF
190 IF j$=my$                        :REMark is it rerun an SBASIC daughter?
200     do SBASIC daughter re-initialisation
210     j$=""                        :REMark no further action required
220 END IF
230 IF j$<>""                        :REMark must be compiled!
240     do compiled BASIC initialisation
250 END IF
```

Within the initialisation code for SBASIC the DEVTYPE function may be used to determine whether a channel is open.

This returns an integer value of which only the most significant (the sign bit) and least significant two bits are set. To ensure future compatibility, nothing should be assumed about the other bits.

The value returned will be negative if there is no channel open. Otherwise bit 0 indicates that it will support window operations (i.e. it is a screen device), bit 1 indicates that it will support file positioning operations (i.e. it is a file).

```
100 a%=DEVTYPE(#3)                   :REMark find the type of device open as #3
110 IF a%<0:PRINT #3, "not open"     :REMark negative is not open
120 SELECT ON a% && %11              :REMark ensure we only look at bits 0 and 1
130     =0:PRINT "#3 is a purely serial device"
140     =1:PRINT "#3 is a windowing device"
150     =2:PRINT "#3 is a direct access (filing system) device"
160     =3:PRINT "#3 is totally screwed up"
170 END SELECT
```

6 Error Reporting and Statement Numbering

SBASIC will usually report error in the form:

At line 250:3 end of file

The number after the colon is the statement number within the line.

N.B. SBASIC generates a small number of additional statements (jumps round DEF PROCs, jumps to END SELect before each ON and END statements on inline clauses) which are not visible in the SBASIC program. If you like piling up structures and statements into a single line, you may find that the statement number in the error report is larger than you would expect!

7 Extended SuperBASIC Commands and Functions

7.1 LOAD, LRUN, MERGE and MRUN: LOAD, LRUN, MERGE and MRUN have been extended to accept Liberation Software's _SAV file format. In addition, if the filename supplied is not found, SBASIC will try first with _BAS and then _SAV added to the end of the filename.

7.2 SAVE and SAVE_O: If no filename is given, the name of the file that was originally loaded will be used (if necessary substituting _BAS or _SAV at the end). The file will be saved with a version number one higher than the file version when it was LOADed. (Repeated SAVEs do not, therefore, keep on incrementing the version number). If a filename is given, the version number is set to 1.

7.3 QLOAD and QLRUN: The extension of the SBASIC LOAD command makes the real QLOAD and QLRUN commands (which require a copy or near copy of QDOS ROMs to function at all) nearly redundant. QLOAD and QLRUN are implemented in SBASIC as versions of LOAD and LRUN that ensure that there is a _SAV at the end of the filename.

7.4 QMERGE, QMRUN, QSAVE and QSAVE_O: These are versions of MERGE, MRUN, SAVE and SAVE_O which work with _SAV files.

If there are 4 SBASIC programs in the data default directory called FRED, JOE, ANNE and CLARA with either _BAS or _SAV at the end of the names:

FRED
JOE_BAS
ANNE_SAV
CLARA_BAS
CLARA_SAV

QLOAD fred	Fails as there is no FRED_SAV
LOAD fred	Loads FRED
SAVE	Saves the program as FRED
QSAVE	Saves the program as FRED_SAV (quickload format)
SAVE junk_bas	Saves the program as JUNK_BAS
QSAVE	Saves the program as JUNK_SAV(quickload format)
MERGE joe	Merges the file JOE_BAS into the program
MERGE anne	Quick merges the file ANNE_SAV into the program
SAVE	Saves it as JUNK_BAS (MERGE does not change the name)
LOAD clara	Loads CLARA_BAS
QLOAD clara	Quick loads CLARA_SAV
LOAD clara_sav	Also quick loads CLARA_SAV

7.5 LBYTES, SBYTES, SBYTES_O, SEXEC and SEXEC_O:

All accept a channel number in place of a name. This can improve efficiency.

```
nc=FOP_IN("file")          Open file once only
base=ALCHP(FLEN(#nc))      ... to allocate a bit of heap
fdt=FUPDT(#nc)             ... get the update date
LBYTES #nc,base            ... and load it
CLOSE #nc
```

7.6 EPROM_LOAD: The EPROM_LOAD(filename) command is a special trick for loading the image of a QL EPROM cartridge. Most EPROM cartridges are programmed so that the cartridge may be at any address. Some require to be at exactly $C000, the QL ROM port address. The first time the command is used after reset, the EPROM image will be loaded at address $C000. Subsequent images may be loaded at any address. A fussy EPROM image must, therefore, be loaded first. An EPROM image file must not be longer than 16K bytes.

To make an EPROM image, put the EPROM cartridge (for example the Prospero PRL cartridge) into your QL and turn on. SBYTES the image to a suitable file with the magic numbers 49152 ($C000) for the base address and 16384 (16K bytes) for the length:

```
SBYTES "flp1_PRL",49152, 16384      Save Prospero PRL image
```

On your SMSQ machine copy the file to your hard disk and add the EPROM_LOAD statement to your "Boot" file:

```
EPROM_LOAD "win1_PRL"               Load Prospero PRL image
```

7.7 PEEK$ POKE$: PEEK$ (address,number of bytes) returns a string with the number of bytes starting from address. The bytes need not, of course, be text.

POKE$ (address,string) pokes the bytes of the string starting from the address.

PEEK$ and POKE$ can be used for copying memory.

```
a$=PEEK$ (Base1,1000)       Peek 1000 bytes from address Base1
POKE$ Base2,a$              ... and poke them back to Base2
```

PEEK$ and POKE$ can accept all the extended addressing facilities of PEEK and POKE. Indeed, POKE$ is identical to POKE which can now which can now accept string parameters.

7.8 PEEK, PEEK_W, PEEK_L, POKE, POKE_W, POKE_L

The standard PEEK functions and POKE procedures have been extended to provide compatibility with the Minerva versions. There are three main changes:

1. The address may be specified relative to the base of the system variables or the (current)

SBASIC variables.

2. The contents of the memory at the address may itself be used as a base address with a second value providing an offset for this address.

3. More than one value may be POKEd at a time.

- For POKE_W and POKE_L the address may be followed by a number of values to poke in succession.

- For POKE the address may be followed by a number of values to poke in succession and the list of values may include strings. If a string is given, all the bytes in the string are POKEd in order. The length is not POKEd.

7.8.1 Absolute PEEK, POKE: The standard forms of PEEK and POKE are supported even though the use of PEEK and POKE is best regarded as a from of terrorism.

```
a=RESPR(2000)
LBYTES myfile,a          Load myfile
PRINT PEEK(a)            Prints the value of the byte of myfile
POKE_L a+28,DATE,0       Set the 28th to 35th bytes to the DATE (4 bytes) and 4 zeros
POKE a+8,0,6,'My_Job'    Set the standard string (work length followed by the chars)
```

7.8.2 Peeking and Poking in the System Variables: If the first parameter of the peek or poke is preceded by an exclamation mark, then the address of the peek or poke is in the system variables or referenced via the system variables. There are two variations: direct and indirect references.

- For direct references the exclamation mark is followed by another exclamation mark and an offset within the system variables.

- For indirect references the exclamation mark is followed by the offset of a pointer within the system variables, another exclamation mark and an offset from that pointer.

```
ramt=PEEK_L(!!$20)       Find the top of RAM $20 bytes on from the base of
                         sysvars
POKE_W !!$8e,3           Set the auto-repeat to 3

job1=PEEK_L(!$68!4)      Find the base address of Job 1 (4 on from base of Job
                         table)

POKE !$B0!2, "WIN"       Change the first 3 characters of DATA_USE to WIN
```

There is slightly more parameter checking than in the Minerva versions. Nevertheless, errors and deliberate abuse are not likely to be detected and may have different effects on SMSQ and Minerva.

7.8.3 Peeking and Poking in the SBASIC Variables: If the first parameter of the peek or poke is preceded by a backslash, then the address of the peek or poke is in the SBASIC variables or referenced via the SBASIC variables. There are two variations - direct and indirect references.

- For the direct references the backslash is followed by another backslash and an offset within the SBASIC variables.

- For indirect references the backslash is followed by the offset of a pointer within the SBASIC variables, another backslash and an offset from that pointer.

```
dal=PEEK_W(\\$94)          Find the current data line number

n6=PEEK_W(\$18\2+6*8)      Find the name pointer for the 6th name in the name table
nl6=PEEK(\$20\n6)          ... and the length of the name
n6$=PEEK$(\$20\n6+1,n16)   ... and the name itself
```

7.9 BPUT BGET: BPUT will accept string parameters to put multiple bytes. BGET will accept a parameter that is a sub-string of a string array to get multiple bytes.

```
BPUT #3,27,'R1'                Put ESC R 1 to channel #3
DIM a$(10):BGET #3,a$(1 TO 6)  Get 6 bytes from #3 to a$
```

7.10 ATAN: The ATAN function has been extended to provide a 4 quadrant result by taking two parameters. If x is greater than 0, ATAN(x,y) gives the same result as ATAN(y/x). Otherwise it returns values in the other quadrants (>PI/2 and <-PI/2).

7.11 PROT_DATE: Where the system has a separate battery backed real time clock, the date is read from the clock when the system is reset. Thereafter, the clock is kept up to date by the SMSQ timer. (Thus the impressive speed gains made by some accelerator software: slowing the clock down by disabling interrupts can do wonders for your benchmark timings.)

In general, the system real time clock is updated whenever you adjust or set the date. As some QL software writers could not resist the temptation of setting the date to their birthday (or other inconvenient date) this can play havoc with your file date stamps, etc..

PROT_DATE(0 or 1) is used to protect (1) or unprotect (0) the real time clock. If the real time clock is protected, setting the date affects only SMSQ's own clock, the real time will be restored the next time the computer is reset.

```
PROT_DATE 1    protect the RTC (should never by required)
PROT_DATE 0    unprotect the RTC (normal)
```

7.12 VER$: The VER$ function has been extended to take an optional, (Minerva compatible) parameter. If it is non zero, information is taken from the OS call for system information. Otherwise, the normal SBASIC version (HBx) is returned.

PRINT VER$	prints HBA (or later SBASIC version ID)
PRINT VER$(0)	also prints HBA (or later SBASIC version ID)
PRINT VER$(1)	prints 2.xx

With a negative parameter, VER$ does not return a version at all, but returns a fairly arbitrary choice of information.

PRINT VER$(-1)	print the Job ID (0 for initial SBASIC)
PRINT VER$(-2)	print the address of the system variables (163840), WHY?

8 Multiple Copies of SBASIC

There never was much of a problem getting multiple copies of SuperBASIC to run under QDOS. There is even less of a problem getting multiple copies of SBASIC to run under SMSQ. The problem was always what to do with the windows.

SBASIC has four distinct forms:

1. Job 0 is the "guardian" of SBASIC extensions, permanent memory allocation and channel # 0.

2. SBASIC "daughter jobs" may be created with the SBASIC command. These may be created with the same set of 3 windows as the initial Job 0 windows. Alternatively, they may be created with a single channel #0 or even no windows open at all.

3. SBASIC source files (ending in _bas) may be executed by EX, EXEC, EW or EXEC_W.

4. SBASIC may be invoked as a Thing which may either operate within the context of an invoking Job or, once set up, operate as an independent daughter Job.

8.1 SBASIC Daughter Jobs: Having a number of SBASIC jobs which completely cover each other may not be very useful. SBASIC daughter jobs may, therefore, either be created either with the full set of standard windows (in which case they all overlap) or they may be created with only one small window (#0).

The SBASIC command, which creates SBASIC daughter jobs, has an optional parameter: the x and y positions of the window #0 in a one or two digit number (or string).

- If no parameter is given, the full set of standard windows will be opened.

- Otherwise, only window #0 will be opened: 6 rows high and 42 Mode 4 characters wide within a 1 pixel border (total 62x256 pixels).

- If only one digit is given, this is the SBASIC "row" number: row 0 is at the top, row 1 starts at screen line 64, row 4 is just below the standard window #0.

- If two digits are given, this is the SBASIC "column,row" (x,y) position: column 0 is at the left, column 1 starts at 256 pixels in from the left.

SBASIC	create an SBASIC daughter with the 3 standard windows
SBASIC 1	create an SBASIC daughter with just channel #0 in row 1
SBASIC 24	create an SBASIC daughter to the right of and below the standard windows (an 800x600 display is required)

Because it is quite normal for an SBASIC job to have only #0 open, all the standard commands which default to window #1 (PRINT, CLS, etc.) or window #2 (ED, LIST, etc.) will default to window #0 if channel #1 or channel #2 is not open. This may not apply to extension commands.

If you have a screen larger than 512x256 pixels, it is useful to be able to reposition the SBASIC windows. The TK2 WMON and WTV commands have been extended to take an extra pair of parameters: the pixel position of the top left hand corner of the windows. If only one extra parameter is given, this is taken to be both the x and y positions.

WMON 4,50	reset windows to standard monitor layout displaced 50 pixels to the right and 50 pixels down

If the mode is omitted, the mode is not changed, and, if possible, the contents are preserved and the outline (if defined) is moved.

WMON ,80,40	reset windows to standard monitor layout displaced 80 pixels to the right and 40 pixels down, preserving the contents

A border has been added to window #0 to make it clearer where an SBASIC Job is on the screen.

8.2 JOB_NAME: The procedure JOB_NAME(job name) can be used to give a name to an SBASIC Job. It may appear anywhere within a program and may be used to reset the name whenever required. This command has no effect on compiled BASIC programs or Job 0.

JOB_NAME Killer	sets the Job name to "Killer"
JOB_NAME "My little Job"	sets the Job name to "My little Job"

8.3 Executing SBASIC Programs: SBASIC program files (ending in _BAS, _bas, _SAV or _sav) may be executed using the EX (EXEC) and EW (EXEC_W) commands.

EX my_little_prog_bas executes the SBASIC program "my_little_prog_bas"

Just as for "executable" programs, if file or device names (or channels) are given after the program name, the first file device or channel will be #0 within the program, the second will be #1, etc.. A simple program for "uppercasing" could be:

```
100 JOB_NAME UC
110 REPeat
120     IF EOF(#0):QUIT
130     BGET #0,a%
140     SELect ON a%
150       =97 TO 122:BPUT #1,a%^^32
160       =REMAINDER: BPUT #1,a%
170     END SELect
180 END REPeat
```

Saved as "uc_bas", this can be used for printing a file in upper case:

EX uc_bas,any_file,par

It can also be used as a filter to uppercase the output of any program sending its output to the "standard output".

EX my_prog TO uc_bas,par

The command QUIT should be used to get rid of an SBASIC job whether it has been created by the SBASIC command, EX or any other means.

8.3.1 Channel #0: There are some oddities in the handling of channel #0 which have been introduced to make the use of SBASIC a little easier.

- On normal completion of a program, if #0 is not open, SBASIC will die naturally. If #0 is open, SBASIC will wait for a command.

- In case of error, if #0 is not open, a default window #0 will be opened for the error message.

- Likewise, if an operation is requested on a default channel (#0, #1 or #2) and neither the default channel or #0 are open, a default window #0 will be opened for the operation.

8.4 SBASIC and Resident Extensions: Resident extensions linked into Job 0 (the initial SBASIC) are available to all SBASIC jobs. If extension procedures and functions are linked into other SBASIC Jobs (using LRESPR), they are local to those Jobs and will be removed when the Jobs die or are removed.

Note that, because or this feature, LRESPR cannot be used from a Job, other than Job 0, to load files which include system extensions (i.e. MENU_REXT, QTYP, etc.).

8.5 SBASIC Executable Thing: The SBASIC executable Thing is called "SBASIC". The provision of an SBASIC executable Thing enables the diehard QDOS fanatic to go well beyond the facilities provided by the SBASIC and EX commands. Depending on how it is invoked, SBASIC can execute independently of the invoking program, or it may take its channels and program from the invoking program.

On being invoked, SBASIC expects to find some channel IDs and a string on the stack (standard QDOS conventions). Because, however, SBASIC requires some BASIC source code in order to be able to execute, the treatment of these channel IDs and the string on the stack are slightly unconventional.

- If SBASIC is invoked without any channel IDs on the stack, SBASIC will behave either as a normal SBASIC interpreter, with the standard set of windows, or as an interpreter with no windows initially opened.

- If the string on the stack is null, the standard set of windows is opened and SBASIC waits for a command. (This is what happens when you give an SBASIC command without parameters or when you start SBASIC from the QPAC2 EXEC menu without a command string.)

- If the string on the stack is not null, no windows are opened and the string is treated as a command line. (This is what happens when you start SBASIC from the QPAC2 EXEC menu after specifying a command string.)

- If there are one or more channel IDs on the stack, SBASIC will normally treat the first ID as the SBASIC source program file, the next ID as channel #0, the next ID as channel #1 and so on. The string defines the initial value of the cmd$ variable within the SBASIC program. (This is what happens when EX executes an SBASIC program.)

- There is a special "trick" for setting up an SBASIC program with just window #0 open. The x,y coordinates of the top left hand corner of the required window #0 are complemented and put on the stack in place of the channel ID.

- If there is only one channel ID on the stack, and this is a "false" ID (i.e. the ID is negative), a 6 line by 42 column channel #0 is opened with the origin at NOT the MSW (x) and NOT the LSW (y) of the false ID. The string is treated as a command line. (This is what happens when you give an SBASIC command specifying the position of window #0.)

- If there are two channel IDs on the stack and the second is a "false" ID, the first is used as the SBASIC program source file and the second ID is used to define window #0. The string defines the initial value of the cmd$ variable. (This could be useful.)

8.6 The SBASIC Interface Things: Two interface Things are provided for the interface to the SBASIC extension which are compatible with two of the established executable program interfaces. The first is called "SBAS/QD" which provides a QD5 compatible F10 interface (Jochen Merz). The second is a "FilenInfo" Thing (Wolfgang Lenerz) which recognises and executes files starting with _sav or _bas.

If QD (Version 5 or later) is configured to use the SBAS/QD Thing, then you can create (line numbered or unnumbered) SBASIC programs with QD and execute them by pressing F10 (shift F5). QD may be temporarily configured to do this by executing it with the appropriate command string.

EX QD;'\T SBAS/QD' Execute QD using SBAS/QD Thing

The FileInfo Thing is used by the QPAC2 Files Menu (amongst others) to determine how to "Execute" a file. With the default FileInfo Thing incorporated into SMSQ, files ending with _sav or _bas may be executed directly from the Files menu and any other utility program which uses the FileInfo Thing.

9 Input Line Editing

The range of standard input line editing keystrokes is now much wider and has been made consistent for INPUT and ED.

Key	With	Operation
Left arrow		move left on character
Right arrow		move right one character
TAB	SHIFT	move left eight characters
TAB		move right eight characters
Left arrow	SHIFT	move left one word
Right arrow	SHIFT	move right one word
Left arrow	ALT	move to start of line
Right arrow	ALT	move to end of line
Left arrow	CTRL	delete left one character
Right arrow	CTRL	delete right one character
Left arrow	CTRL SHIFT	delete left one word
Right arrow	CTRL SHIFT	delete right one word
Left arrow	CTRL ALT	delete to start of line
Right arrow	CTRL ALT	delete to end of line
Down arrow	CTRL	delete whole line

Some keyboards have Delete and Backspace keys:

Key	With	Operation
Backspace		delete left one character
Delete		delete right one character
Backspace	SHIFT	delete left one word
Delete	SHIFT	delete right one word
Backspace	ALT	delete to start of line
Delete	ALT	delete to end of line

10 Language Facilities

SMSQ/QXL incorporates several language variations and extra variations may be added "at run time".

10.1 Language Specification: A language may be specified either by an international dialling code or an international car registration code. These codes may be modified by the addition of a digit where a country has more than one language.

Language Code	Car Registration	Language and Country
33	F	French (in France)
44	GB	English (in UK)
49	D	German (in Germany)

10.2 Language Control Procedures: There is a set of procedures and functions which allow the language of the messages, the keyboard layout and the printer translate table to be set. Where a language is to be specified, the parameter may be an integer value (the telephone dialling code), a string (the car registration letters), a variable or expression which yields an integer or string result, or a variable name.

It is not necessary for the car registration letters to be in upper case.

10.2.1 LANG_USE: The language of the messages is set by the LANG_USE command. This sets the OS language word, and then scans the language dependent module list selecting modules and filling in the message table.

LANG_USE 33	set language to French
LANG_USE D	set language to German
LANG_USE 'g'&'b'	set language to English

WARNING: if you assign a value to a variable, then you will not be able to use that variable name to specify the car registration letters.

D=33:LANGUAGE D	set language to French (dialling code 33) rather than German

10.2.2 LANGUAGE LANGUAGE$: The LANGUAGE and LANGUAGE$ functions are used to find the currently set language, or to find the language that would be used if a particular language were requested. They can be used to convert the language (dialing code) into car registration and vice versa.

PRINT LANGUAGE	the current language
PRINT LANGUAGE$	the car registration of the current language
PRINT LANGUAGE(F)	the language corresponding to F
PRINT LANGUAGE$(45)	the car registration corresponding to 45
PRINT LANGUAGE(977)	the language that would be used for Nepal

10.2.3 KBD_TABLE: The keyboard tables are selected by the KBD_TABLE command.

KBD_TABLE GB	keyboard table set to English
KBD_TABLE 33	keyboard table set to French

Private keyboard tables may also be loaded.

i=RESPR(512):LBYTES "kt",i:KBD_TABLE i keyboard table set to table in "kt"

For compatibility with older drivers, a "private" keyboard table loaded in this way should not be prefaced by a flag word.

10.2.4 TRA: The SBASIC TRA command differs very slightly in use from the QL JS and MG TRA. The differences are quite deliberate and have been made to avoid the unfortunate interactions between functions of setting the OS message table and setting the printer translate tables. If you only wish to set the printer translate tables, the only difference is that TRA 0 and TRA 1 merely activate and disactivate the translating. They do not smash the pointer to the translate tables if you have previously set it with a TRA(address) command.

If you wish to change the system message tables, then the best way is to introduce a new language. This is done by LRESPRing suitable message tables.

Language dependent printer translate tables are selected by the TRA (1, lang) command. If no language code or car registration code is given, the currently defined language is used.

Language independent translate tables are set by the TRA (n) command where n is a small odd number.

Private translate tables are set by the TRA (addr) command where addr is the address of a table with the special language code $4AFB.

TRA 0	translate off, table unchanged
TRA 0,44	translate off, table set to English
TRA 0,F	translate off, table set to French
TRA 1	translate on, table unchanged
TRA 1,GB	translate on, table set to English
TRA 1,33	translate on, table set to French
TRA 3	translate on, table set to IBM graphics

A=RESPR(512):LBYTES "tratab",:TRA A	translate on, table set to table in "tratab"

To use the language independent tables, your printer should be set to USA (to ensure that you have all the # $ @ [ ] { } \ ^ ~ symbols which tend to go missing if you use one of the special country codes (thank you ANSI), and select IBM graphics codes as appropriate.

For the IBM tables, QDOS codes $C0 to $DF are passed through directly and QDOS codes $E0 to $EF are translated to $B0 to $BF to give you all the graphic characters in the range $B0 to $DF. QDOS codes $F0 to $FF are passed through directly to give access to the odd characters at the top of the IBM set.

10.3 BAUD Command:

For the QXL, the standard BAUD command mimics the QL BAUD command.

BAUD 4800	Set SER1 and SER2 to 4800 baud

Both the SuperBASIC BAUD command and the OS baud trap have been extended to support independent baud rates for each serial port.

BAUD 1,19200	Set SER1 to 19200 baud
BAUD 2,0	Set SER2 to 153600 baud

11 Virtual Devices

Virtual devices are not associated with any physical hardware. NUL devices are complete dummies (very useful for benchmarking: SMSQ/QXL has one of the fastest, if not the fastest, fully functional NUL device in the world). PIPEs are buffers for storing information or passing it from one task to another. The PIPE is double ended: what goes in one end, comes out the other in the same order (FIFO - first in first out).

11.1 NUL Device:

The NUL device my be used in place of a real device. The NUL device is usually used to throw away unwanted output. It may, however, be used to provide dummy input or to force a job to wait forever. There are five variations.

NULP waits (forever or until the specified timeout) on any input or output operation.

NUL, NULF, NULZ and NULL ignore all operations (the output is thrown away).

NUL, NULF, NULZ and NULL return a zero size window in response to window information requests. Pointer Information calls (IOP.PINF, IOP.RPTR) return an invalid parameter error.

NUL is an output only device, all input operations return an invalid parameter error.

NULF emulates a null file. Any attempt to read data from NULF will return an End of File Error as will any file positioning operation. Reading the file header will return 14 bytes of zero (no length, no type).

NULZ emulates a file filled with zeros. The file position can be set to anywhere. Reading the file header will return 14 bytes of zero (no length, no type).

NULL emulates a file filled with null lines. The file appears to be full of the newline character (CHR$(10)). The file position may be set to anywhere. Reading the file header will return 14 bytes of zero (no length, no type).

11.2 PIPE Device: There are two variations on the PIPE driver: named and unnamed pipes. Both of these are used to pass data from one program to another. Unnamed pipes cannot be opened with the SBASIC OPEN commands but are opened automatically by the EX and EW commands when these are required to set up a "production line" of Jobs. Whereas, if a pipe is identified by name, any number of Jobs (including SBASIC) can open channels to it as either inputs or outputs.

If, using named pipes, matters become confused, then that is a problem for the Jobs themselves. This is not as bad as it sounds. Unlike other devices, named pipes transfer multiple byte strings atomically unless the pipe allocated is too short to hold the messages. This means that provided the messages are shorter than the pipe, many jobs can put messages into a named pipe and many jobs can take messages out of a named pipe without the messages themselves becoming scrambled.

If a PIPE is shared in this way, there are two simple ways of ensuring that the messages are atomic. The first, using fixed length messages, is not available to SBASIC programs. The second, using "lines" terminated by the newline character, works perfectly. N.B. the standard PRINT command will not necessarily send a line as a single string for each item output.

PRINT #3,a$\b$	Bad, sends 4 strings: the newline are separate
PRINT #3,a$&CHR$ (10):	Good, sends 1 string, including the newline
INPUT #4,b$	Good, reads a single line from the pipe

Named pipes should be opened with OPEN_NEW (FOP_NEW) for output and OPEN_IN (FOP_IN) for input. A named pipe is created when there is an open call for a named pipe which does not exist. It goes away when there are no longer any channels open to it and it has been emptied.

As well as the name, it is possible to specify a length for a named pipe. If the pipe already exists, the length requested is ignored.

OPEN_NEW #4,PIPE_xpf	Open named output pipe of default length (1024 bytes)
OPEN_NEW #5,PIPE_frd_2048	Open named output pipe of length 2048 bytes
OPEN_IN #6,PIPE_xfr	Open named input pipe

11.3 DEV - A Virtual Filing System Device: DEV is a defaulting device that provides up to 8 default search paths to be used when opening files. As it was designed to be dumped on top of QDOS it is not very clean, but, equally, it is reasonably efficient.

Each DEV (DEV1 to DEV8) device is a pseudonym for a real filing system device or directory on a filing system device.

Files on a DEV device can be OPENed, used and DELETEd in the same way as they can on a real device.

11.3.1 DEV_USE: Each DEV device is defined using the DEV_USE (number,name,next) which specifies the number of the DEV device, the real device or directory and the next device in the chain.

DEV_USE 1,ram1_	DEV1_ is equivalent to ram1_
OPEN #3,dev1_f1	opens ram1_f1
DEV_USE 2,flp1_ex_	DEV2_ is equivalent to flp1_ex_
OPEN #3,dev2_f1	opens flp1_ex_f1
DEV_USE 3,win1_work_new	DEV3_ is equivalent to win1_work_new
OPEN #3,dev3_f1	opens win1_work_newf1
DELETE dev3_junk	deletes win1_work_newjunk

Note that, unlike the defaulting commands PROG_USE and DATA_USE, the underscore at the end of the real device or directory is significant.

There is a neat variation on the DEV_USE call which enables you to set up default chains. If you put a "next" number at the end of the DEV_USE command, this will be taken as the DEV to try if the open fails. This next DEV can also chain to another DEV. You can even close the chain: the DEV driver will stop chaining when it has tried all the DEVs in the chain.

```
DEV_USE 1,ram1_,3          DEV1_ is equivalent to ram1_, next is DEV3_
DEV_USE 2,flp1_ex_,1       DEV2_ is equivalent to flp1_ex_, next is DEV1_
DEV_USE 3,win1_work_,2     DEV3_ is equivalent to win1_work_, next is DEV2_

LOAD dev1_anne             will try        ram1_anne (DEV1)
                           then            win1_work_anne (DEV3)
                           and finally     flp1_ex_anne (DEV2)

LOAD dev2_anne             will try        flp1_ex_anne (DEV2)
                           then            ram1_anne (DEV1)
                           and finally     win1_work_anne (DEV3)
```

Note that DELETE only operates on the DEV specified, it does not chain.

A DEV default may be cleared by giving no name.

```
DEV_USE 2          clear definition for DEV2_
```

11.3.2 DEV_LIST: DEV_LIST(channel) lists the currently defined DEVs in the specified channel (default #1).

```
DEV_LIST           lists the current DEVs in #1
DEV_LIST #2        lists the current DEVs in #2
```

11.3.3 DEV_USE$ DEV_NEXT: The DEV_USE$(number) function returns the usage for the specified DEV. The DEV_NEXT(number) function returns the next DEV after the specified DEV.

```
PRINT DEV_USE$(3)          prints the usage for DEV3_
PRINT DEV_NEXT(1)          prints the next DEV in the chain after DEV1_
```

11.3.4 Interaction between DATA_USE, PROG_USE & DEV: If you are going to use the DEV defaults, it makes sense to set the DATA_USE and PROG_USE defaults to use DEV, and when moving from directory to directory change the DEV definition rather than the DATA_USE.

```
DATA_USE dev1_             data default directory is DEV1_
DEV_USE 1,flp2_myprogs_    ... which is myprogs_ on FLP2_
PROG_USE dev2_             programs from DEV2_
DEV_USE 2,flp1_ex_,1       ... which is flp1_ex_ or my data default!
```

12 SMSQ/QXL Trouble Shooting

Q1: I get a "not found" error message when trying to write a file to a DOS format diskette.

A1: There are three problems to watch out for.

The first is that it is obligatory to create directories (if required) on a DOS format disk before you put files into them. You cannot just use any name you like as you have been accustomed to do on QDOS format disks.

The second is that the filing system automatically converts underscores to directory separators: a filename "MY_FILE" cannot be distinguished from a file "FILE" in a directory "MY".

The third is that the filing system does not automatically attempt to convert files which end in, for example, "_bas" to files ending in ".bas".

Q2: Files which I intended to write to a DOS format floppy disk were copied instead to my data default directory with "FLP1_" in front of the name.

A2: The same problem as above.

Q3: When I try to QMON a file, I end up tracing Job 0 in a rather bizarre way.

A3: Unlike SuperBASIC, SBASIC initialises all variables to zero or null string.

In order to distinguish a name (which would be a filename) from a number (which would be a job number) QMON, and a small number of other programs, made the assumption that a file name would not have a value. This is not necessarily true for SuperBASIC, it is never true for SBASIC.

Either put the file name in quotes or upgrade your QMON.

Q4: I have some software which works in the initial SBASIC but does not work when I try to use it from an SBASIC daughter.

A4: The initial SBASIC is 99.9% compatible with SuperBASIC. SBASIC daughter Jobs are only 99% compatible with SuperBASIC. In particular they are not Job 0 and the channel IDs for #0, #1 and #2 are not $00000, $10001 and $20002.

Some software cannot cope with this difference. Old versions of QMON cannot, but old versions of JMON can. Turbo cannot, QLiberator can.

Either update the software or use this software in the initial SBASIC only.

Q5: I get the message "unknown error" instead of a proper error message when I use the SBAS/QD F10 Thing.

A5: Some of the SBASIC error messages are longer (hopefully more helpful) than the old SuperBASIC error messages. Older versions of the Menu extensions cannot cope with these long messages. Update your MENU_REXT and WMAN.

Q6: When I try to use the SuperBASIC channel table from another Job, I find that it is empty.

A6: As you can have many SBASIC Jobs, you can have many SBASIC channel tables: one in each set of SBASIC variables. From within an SBASIC Job, these look just the same as the SuperBASIC channel table.

The tricks that can be used with QDOS to find the SuperBASIC variables area will, in SMSQ, find a dummy variables area which holds only the global name table. This is the only part of the SuperBASIC environment which is common to all SBASIC Jobs.

To find any other part of a SuperBASIC variables area from another Job, you must define which copy of SBASIC you wish to poke about in. To do this you need to go to supervisor mode and find the value of A6 for the particular SBASIC Job you are about to interfere with: the channel table and most other parts of the SuperBASIC variables area will be found at their usual offsets from A6.

Miracle Systems Ltd

SUPER GOLD CARD SUPPLEMENT

20 Mow Barton, Yate, Bristol, BS17 5NF, UK
Telephone/Fax: (01454) 883602

MIRACLE SYSTEMS LTD Super Gold Card Supplement

(NOTE: If you are using a Gold Card please skip this supplement.)

OVERVIEW

The Super Gold Card brings true 32-bit processing to the QL for the first time together with several other benefits. These are:

68020 microprocessor running at 24MHz for high speed
3968K of 32-bit, no wait state memory seen by QDOS
Parallel port for fast graphics printing
Two disk drive connectors for four drive capability
Battery backed clock with crash protection
Automatic booting option for unattended operation
Automatic enabling of Toolkit II option
External regulated 5V input connector to allow easy connection of switch mode power supply

Fitting the Super Gold Card: The Super Gold Card contains static sensitive devices and should be kept in the anti-static bag supplied when not plugged into the QL. Other types of anti-static bag could cause the clock battery to discharge.

Make sure that the power to the QL is switched off. Remove the cover at the left hand end of the QL. Very carefully plug the Super Gold Card into the now exposed expansion port. The gold coloured heatsink should just touch the QL's case top when the Super Gold Card is fully plugged in. Power may now be applied.

The Super Gold Card has all the functionality of the Gold Card and all references in the manual to Gold Card also refer to the Super Gold Card.

Connecting the disk drives: There are two disk drive connectors on the Super Gold Card. If you have a double drive it should be plugged into the upright connector, i.e the one nearest the QL, and the drives are FLP1_ and FLP2_ as expected.

A second double drive may be plugged into the connector to the left of the first and the drives become FLP3_ and FLP4_.

The upright connector is wired so that all four drives can be connected to it via a single cable but the third and fourth drive mechanisms must have their jumpers set for DS2 and DS3 respectively. If in doubt please consult the data sheet for the drive mechanism in question.

Details of the disk drive connectors appear later in this supplement.

PARALLEL PORT

To connect the Super Gold Card's parallel port to the printer plug one end of the cable provided into the connector below the gold coloured heatsink and plug the other into the printer. The main advantage of the parallel port over the serial port is high speed printing of graphics but it also simplifies setting up as once the cable connects the Super Gold Card to the printer nothing else is required.

The device name is PAR and in addition to the parameters accepted by the SER device four new options are available. The T option, which is the default, will use an active translation table. The D option does not translate. (The default SDP_DEV uses PARD. See page 10 for more on SDP_DEV.) The A option precedes every line feed, CHR$(10), with a carriage return, CHR$(13). The A option is only necessary if the printed page looks something like this:

This is line one
This is line two
This is line three

The F option sends a form feed, CHR$(12), at the end of a file i.e. when the channel is closed.

The following will print a file ending with a form feed: COPY_N "flp1_myfile" TO "parf"

To print the same file followed by a form feed but with all line feeds preceded by carriage returns: COPY_N "flp1_myfile" TO "paraf"

For compatibility with other parallel port drivers, the name PAR may be followed by an underscore and a number, optionally ending in a K character although this is ignored. For example: OPEN #3,"par_6K"

The PAR_USE command: To allow the parallel port to be used with software that only allows output to SER1 or SER2 a command PAR_USE has been provided. An example is:

```
PAR_USE "ser"
COPY_N "flp1_myfile" TO "ser2"
```

which will send the file to PAR.

If you use QUILL and it currently prints to one of the serial ports just put the line PAR_USE "ser" at the beginning of the boot program and it will print using the parallel port.

PAR_USE also allows the use of extra parameters. For example

```
PAR_USE "ser"
COPY_N "flp1_myfile" TO "ser1f"
```

will print the file to PAR ending with a form feed.

Buffering output to PAR: To print a file using the parallel port using free memory as a buffer enter the following:

```
PAR_USE "lpt"
PRT_USE "par","lpt"
COPY_N "flp1_myfile" TO "par"
```

If you are using a program, for example QUILL, configured to print via SER1 or SER2 and you wish it to print via PAR with a buffer add the following to the start of the BOOT program.

```
PRT_USE "ser","par"
```

See later in the manual for more information on PRT_USE. Details of the parallel port connector appear later.

ENABLING TOOLKIT II & AUTOMATIC BOOTING

If your QL has a Sinclair ROM installed there is the option of simulating the pressing of F1 or F2 as well as invoking TK2_EXT. On a QL fitted with Minerva these commands will only invoke TK2_EXT as all Minervae auto boot.

To invoke TK2_EXT and simulate pressing F1 enter: AUTO_TK2F1
To invoke TK2_EXT and simulate pressing F2 enter: AUTO_TK2F2

To disable automatic booting or if you have software that does not work if Toolkit II has been enabled enter: AUTO_DIS

FLP JIGGLE

The Gold Card and Super Gold Card incorporate a software fix for a problem when accessing Mitsubishi ED disk drives in the form of a number of rapid steps of the drive head. This rapid stepping of the drive head, referred to as a jiggle, when applied to other makes and types of disk drive can result in the drive giving 'not found' or 'bad or changed medium' errors. There is a command to enable or disable this jiggle.

Suppose your system comprises one of our Dual ED Disk Drives for FLP1_ and FLP2_ and some other make of double drive for FLP3_ and FLP4_. To boot up reliably the head jiggle must be enabled so from the command line enter: FLP_JIGGLE 1

FLP3_ and FLP4_ do not require the head jiggle so add the following lines to the start of your boot program:

```
FLP_JIGGLE 3,0
FLP_JIGGLE 4,0
```

If the command is given only one parameter it is memorised and remains in force the next time the QL is powered up or reset. Individual settings are not memorised and have to be set each time the QL is powered up or reset.

MIRACLE SYSTEMS LTD Super Gold Card Supplement

THE CACHE

There is a cache on the Super Gold Card that can increase performance but it can cause problems with programs that modify themselves during execution.

There is no way of knowing whether or not a program is self-modifying so try each program first with the cache off, by typing: CACHE_OFF and then with the cache on, bytyping: CACHE_ON

If the program behaves differently with the cache on, other than going slightly faster, it is a sign that it is self-modifying and should only be run with the cache off.

EVEN MORE SPEED

The QL's hardware allows the screen to be taken from one of two areas in memory. Sinclair ROMs offer no choice but some games ignore the ROM and use the second screen and Minerva offers this as an option. If you do not make use of the second screen it is possible to disable it and gain maximum performance.

Entering the command SCR2DIS disables the second screen. To reenable it enter SCR2EN This command is memorised by the Super Gold Card so it is only necessary to type it once.

EXTERNAL POWER INPUT

If you are using your system with a standard power supply ignore this section. If you wish to power your system from a switch mode power supply 5V may be fed into the Super Gold Card directly via a 2.1mm power connector located under the heatsink. The heatsink should not be removed and the regulator should be left in place.

Wire the plug with +5V on the outside and the centre grounded. A suitable plug is available from Miracle Systems.

ASSEMBLY LANGUAGE

If you are interested in using assembly language on the QL please read this section.

First and foremost the Super Gold Card is for running existing QL programs faster. QL programs are written for the 68000 and there is software on the Super Gold Card to make it look as much as possible like a 68000 system.

Also we recommend that your programs do not make use of special 68020 instructions so that they run on all QL systems. This is a particularly important consideration for any program destined for sale or publication. The golden rule is: assume you are writing for the 68000.

Incidentally, we did not use or need a 68020 assembler in extending the Gold Card software for the Super Gold Card. The only 68020 instruction is the one to write to the cache control register.

Additional information regarding the 68020 is NOT available from Miracle Systems.

The cache from assembler The cache is always on in supervisor mode and should not be turned off. To do so would cause problems for device drivers such as the Microdrives which are written to work at the speed of the 68020 with its cache enabled.

There is no operating system call to alter the cache mode. The CACHE_ON command should be used from SuperBASIC. However, it is possible to turn the cache on from user mode temporarily by entering supervisor mode. Once the program goes back to user mode the cache will be disabled after the next interrupt or TRAP call.

Reading the status register The instruction to read the status register:

MOVE.W SR,<ea>

is privileged on the 68020. For QL program compatibility the Super Gold Card intercepts this privilege violation and emulates the instruction in software.

Exception stack frame The 68020 stacks at least one additional word of information when an exception is generated and consequently expects this additional information when the RTE instruction is executed.

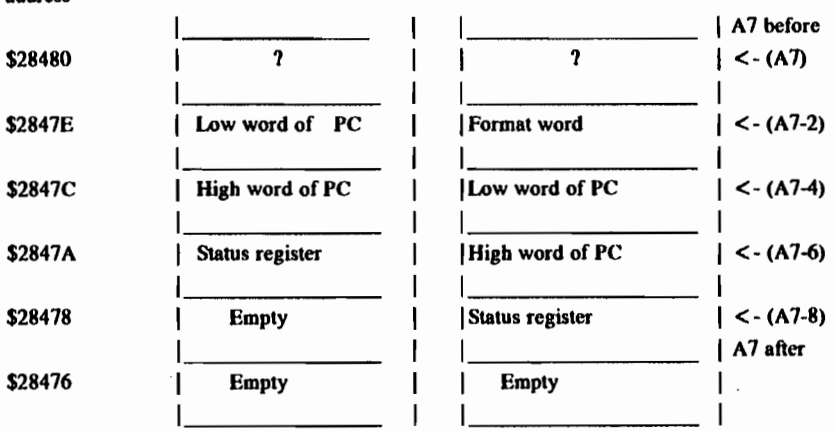
If your programs use the technique of going from user mode into supervisor mode and stacking an address and the status register and then executing an RTE to return to user mode they will cause problems on the Super Gold Card.

If your program entered supervisor mode from user mode only return to user mode by writing to the status register.

User definable exception vectors All the user definable exception vectors that require it are preceded by code that makes the information on the stack look like that generated by the 68000 except that the format word stacked by the 68020 is also there. This need not be a problem as the format word is the first thing stacked as explained by the following example.

Suppose you wish to use the trace mode to invoke your own set of routines. The exception stack frame on the left is as generated on a QL and the one on the right by a QL fitted with the Super Gold Card and not as that generated by the 68020.

The stacked PC is the address of the instruction being traced.



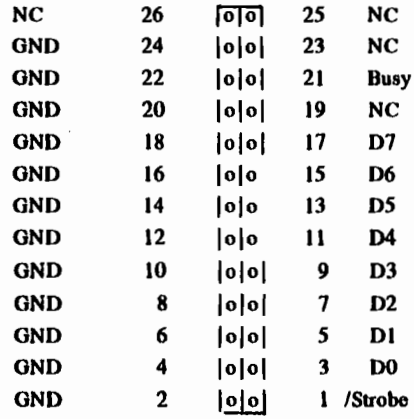
The trace handler might look a bit like this:

```
MOVEM.L   D0/A0,-(A7)      * Save a couple of registers
MOVE.L    2*4+2(A7),A0     * Get address of instruction
MOVE.W    (A0),D0          * Get instruction itself
...                        * Take some action
MOVEM.L   (A7)+,D0/A0      * Restore registers saved
RTE                        * Resume execution of main program
```

The above example would work without modification on the QL with or without a Super Gold Card and know nothing about the extra word on the stack.

PRINTER PORT CONNECTOR DETAILS

View looking into the plug.



MIRACLE SYSTEMS LTD Super Gold Card Supplement

THE EXPANSION CONNECTOR:

The expansion connector on the Super Gold Card is detailed below. Signals marked NC are signals either not required or not provided by the Super Gold Card depending on whether they are inputs or outputs respectively. Below is a view into the Super Gold Card's expansion connector.

Use on SGC				Use on SGC		
VIN	VIN	b 32	a	VIN	VIN	
VIN	VIN	b 31	a	VM12	NC	
NC	EXTINTL	b 30	a	VP12	NC	
IPL1L	IPL1L	b 29	a	SP0	GND	
NC	BERRL	b 28	a	SP1	GND	
NC	IPL0L	b 27	a	DSMCL	NC	
GND	SP3	b 26	a	SP2	GND	
A2	A2	b 25	a	DBGL	NC	
A1	A1	b 24	a	A3	A3	
NC	ROMOEH	b 23	a	A4	A4	
A0	A0	b 22	a	A5	A5	
NC	FC0	b 21	a	A6	A6	
NC	FC1	b 20	a	A7	A7	
NC	FC2	b 19	a	A8	A8	
NC	BLUE	b 18	a	A9	A9	
NC	GREEN	b 17	a	A10	A10	
NC	VPAL	b 16	a	A11	A11	
NC	VSYNCH	b 15	a	A12	A12	
NC	E	b 14	a	A13	A13	
Refresh	CSYNCL	b 13	a	A14	A14	
RESETCPUL	RESETCPUL	b 12	a	RED	NC	
A15	A15	b 11	a	CLKCPU	NC	
GND	BRL	b 10	a	A16	A16	
NC	BGL	b 9	a	A17	A17	
DTACKL	DTACKL	b 8	a	A18	A18	See text
RDWL	RDWL	b 7	a	A19	A19	See text
DSL	DSL	b 6	a	D7	D7	
Same as DSL	ASL	b 5	a	D6	D6	
D0	D0	b 4	a	D5	D5	
D1	D1	b 3	a	D4	D4	
D2	D2	b 2	a	D3	D3	
GND	GND	b 1	a	GND	GND	

Signals A0..A19, D0..D7, DSL and RDWL are high impedance while RESETCPUL is low.

The Super Gold Card can drive peripherals, which it maps into the area from $4C0000 to $4FFFFF, provided that they do not use VPAL, E, BRL, BGL or BERRL. Signals are not synchronised with CLKCPU. Signals A18 and A19 on the expansion connector are both high during accesses to the peripheral area. A18 and A19 are both low during other accesses to the QL.

MIRACLE SYSTEMS LTD Super Gold Card Supplement

DISK DRIVE CONNECTOR DETAILS

View looking into the plug pointing upwards.

Direction	Function			Function	
-	NC	34	|o|o|	33	GND
OUTPUT	SIDE ONE SELEC ECT	32	|o|o|	31	GND
INPUT	READ DATA	30	|o|o|	29	GND
INPUT	WRITE PROTECT	28	|o|o|	27	GND
INPUT	TRACK ZERO	26	|o|o|	25	GND
OUTPUT	WRITE GATE	24	|o|o|	23	GND
OUTPUT	WRITE DATA	22	|o|o|	21	GND
OUTPUT	STEP	20	|o|o	19	GND
OUTPUT	STEP DIRECTION	18	|o|o	17	GND
OUTPUT	MOTOR ON	16	|o|o	15	GND
OUTPUT	DRIVE SELECT 2	14	|o|o|	13	GND
OUTPUT	DRIVE SELECT 1	12	|o|o|	11	GND
OUTPUT	DRIVE SELECT 0	10	|o|o|	9	GND
INPUT	INDEX	8	|o|o|	7	GND
OUTPUT	DRIVE SELECT 3	6	|o|o|	5	GND
-	NC	4	|o|o|	3	GND
-	NC	2	|o|o|	1	GND

View looking into the plug pointing sideways.

Direction	Function			Function	
-	NC	34	|o|o|	33	GND
OUTPUT	SIDE ONE SELECT	32	|o|o|	31	GND
INPUT	READ DATA	30	|o|o|	29	GND
INPUT	WRITE PROTECT	28	|o|o|	27	GND
INPUT	TRACK ZERO	26	|o|o|	25	GND
OUTPUT	WRITE GATE	24	|o|o|	23	GND
OUTPUT	WRITE DATA	22	|o|o|	21	GND
OUTPUT	STEP	20	|o|o	19	GND
OUTPUT	STEP DIRECTION	18	|o|o	17	GND
OUTPUT	MOTOR ON	16	|o|o	15	GND
OUTPUT	DRIVE SELECT 2	14	|o|o|	13	GND
OUTPUT	DRIVE SELECT 1	12	|o|o|	11	GND
OUTPUT	DRIVE SELECT 0	10	|o|o|	9	GND
INPUT	INDEX	8	|o|o|	7	GND
OUTPUT	DRIVE SELECT 3	6	|o|o|	5	GND
-	NC	4	|o|o|	3	GND
-	NC	2	|o|o|	1	GND

Miracle Systems Ltd

GOLD CARD SUPPLEMENT

20 Mow Barton, Yate, Bristol, BS17 5NF, UK
Telephone/Fax: (01454) 883602

The GOLD CARD contains static sensitive devices and should be kept in the anti-static bag supplied when not plugged into the QL. Other types of anti-static bag could cause the clock battery to discharge.

WARNING

The GOLD CARD must not be used on a QL fitted with the Q-Power regulator. If one is fitted to the QL then replace it with the original regulator before plugging in the GOLD CARD.

Connecting Up

Make sure that the power to the QL is switched off. Remove the cover at the left hand end of the QL. Very carefully plug the GOLD CARD into the now exposed expansion port. Only the gold heat sink should be left protuding beyond the left hand extremity of the QL. If disk drives are to be connected plug their cable into the connector on the exposed edge of the GOLD CARD. Power may now be applied. It is recommended that the whole system is powered up at the same time. Alternatively, power up the peripherals first and then switch on the mains to the QL. Do not switch the QL on and off using the low voltage connector at the back of the QL as this can permanently damage the QL, GOLD CARD or both.

After power up the QL does two resets. During the first reset the GOLD CARD checks the first 128K of RAM then reads the contents of the QL's ROMs. Their contents are copied over to the GOLD CARD RAM and patched. A second reset is then run that checks all the memory. To auto-boot from disk make sure that the disk is in FLP1 before the F1/F2 prompt appears.

DISK DENSITIES

The GOLD CARD supports the following densities:

DD - Double Density,	old QL standard, 720K bytes
HD - High Density,	1.44M bytes
ED - Extra High Density,	new QL standard, 3.2M bytes

The capacity that a diskette will be formatted to depends on the diskette and the drive according to the following table:

Diskette: Drive	DD	HD	ED
DD	720k	720k?	720k? (? = unreliable)
HD	720k	1.44M	720k?
ED	720k	1.44M	3.2M

If you bought your disk drives from Miracle Systems you may ignore the section regarding step rates and proceed to the section headed SUBDIRECTORIES.

Disk drive step rate and disks that will not boot: Some older drives may not work correctly with the step rates provided by default on the GOLD CARD. However, there is a way to enable such drives to boot as long as the correct step rate is known. The numbered steps given below assume a dual drive with a step rate of 6 milliseconds (ms) and a start up time of 800 ms.

1. Reset the QL
2. Press F1 or F2.
3. Type TK2_EXT then press ENTER.
4. Put a blank diskette in FLP1_.
5. Put the diskette holding the software to be copied in FLP2_.
6. Type 100 FLP_START 40 then press ENTER.
7. Type 110 FLP_STEP 6 then press ENTER.
8. Type RUN then press ENTER.
9. Type FORMAT FLP1_MIRACLE then press ENTER.
10. Type 120 LRUN FLP1_BOOT1 then press ENTER.
11. Type SAVE BOOT then press ENTER.
12. Type COPY FLP2_BOOT TO BOOT1 then press ENTER.
13. Type WCOPY FLP2_ TO FLP1_ then press ENTER.
14. Press the letter A when requested.
15. When asked to overwrite FLP1_BOOT press N.
16. When both drives' lights are out remove the diskette in FLP2_.
17. Press the reset button.
18. Pressing F1 or F2 should now boot as expected.

FLP STEP in more detail: The step rate is set automatically by the GOLD CARD to be 3ms for an 80 track drive and 6ms for a 40 track. This can be overridden using FLP_STEP. If only one parameter is given its value applies globally, e.g.

FLP_STEP 12

will set the step rate on all drives to 12ms. When two parameters are given the first is the drive number and the second the step rate, e.g.

FLP_STEP 3,6

will set FLP3 to step at 6ms. Repeated seek errors cause the step rate to be slowed.

SUBDIRECTORIES

Creating a subdirectory: The command MAKE_DIR is used to create a new subdirectory. It takes one parameter: the subdirectory filename. As it can return a variety of errors there is also a function to do the same.

operation: FMAKE_DIR which returns the error code or 0.

MAKE_DIR filename or ferr=FMAKE_DIR (filename)

Normal Error Codes

- 7	not found	Medium or drive is not available
- 8	already exists	Already directory/file of that name
- 9	in use	Already directory/file of that name
-15	bad parameter	Device can not handle subdirectories

If there are any files which, by virtue of their names, would belong in the directory being made, then these files will be transfered to the new directory, even if they are open. For example:

MAKE_DIR "FLP2_letters" or MAKE_DIR "FLP2_letters_"

followed by

DIR FLP2_

would show in its output the file "letters ->". The " ->" signifying that "letters" is a subdirectory. Copying a file to "FLP2_letters_bankmanager" would create a new file in the "letters_" subdirectory and

DIR FLP2_letters_

would now show one file: "letters_bankmanager".

To remove a subdirectory, firstly delete its contents then delete the subdirectory itself. COPY and WCOPY deal only with files at the specified directory level. Subdirectories can also be applied to RAM disks. Please note that subdirectories should not be put on disks that are to be used with a TRUMP CARD system.

THE DEV DEVICE

DEV is a generalised default device and is a fudge to enable existing software, such as Quill, Archive, Abacus and Easel to make use of subdirectories. It is not intended as an excuse to write bad software.

As usual, there are up to 8 DEV devices; DEV1 to DEV8. Each DEV device is attached to a particular real device or a particular default directory on a real device. Files on a DEV device can be opened, used, and deleted in the same way as on a real device. Note that DEV definitions are global.

Each DEV is attached to a device by the DEV_USE command.

DEV_USE DevNumber, RealDirectory

An example of DEV USE with QUILL: The following description assumes a version of QUILL configured to take its files from mdv1_ and mdv2_. Floppy disk users with a copy of QUILL reconfigured to access flp should replace occurrences of mdv with flp.

Place the original copy of QUILL in mdv1_ and, after having pressed F1 or F2, put the floppy diskette on which the program is to run in flp1_ and enter these lines of BASIC.

```
TK2_EXT
MAKE_DIR flp1_quill
MAKE_DIR flp1_quill_data
WCOPY mdv1_ TO flp1_quill_  :REM Press A to copy all
```

Now enter the following BASIC program.

```
100 INPUT "which dir [flp1_quill_data_]?"!DataSource$
105 IF DataSource$="":DataSource$="flp1_quill_data_"
110 DEV_USE 1,FLP1_quill_
120 DEV_USE 2,DataSource$
130 DEV_USE mdv
140 EXEC_W mdv1_quill
150 DEV_USE  :REM Clear the DEV setting after quitting

SAVE flp1_quill_boot  :REM Key Y to overwrite old boot
```

In future QUILL can be run by entering: "LRUN flp1_quill_boot"

Files will then load from the specified directory. The subdirectory in which the QUILL _doc files are to be found is first requested. If ENTER is pressed the program assumes files are located in a subdirectory called DATA_ in the QUILL_ subdirectory. Otherwise enter the name of the device followed, optionally, by the subdirectory holding the files.

The same process can be employed with Archive, Abacus and Easel as there is plenty of space even on a double density diskette for all four programs. The fact that they are in different subdirectories means they can all have their own PRINTER_DAT.

Running INSTALL BAS using DEV: INSTALL_BAS was written with Microdrives in mind but the listing given below shows how this can be circumvented. Assuming you have just copied the contents of the QUILL Microdrive to flp1_quill_, enter this:

```
NEW    :REM Clear out the old program

100 DEV_USE 1,flp1_quill_
110 DEV_USE 2,flp1_quill_
120 DEV_USE mdv
130 LRUN mdv1_install_bas
```

SAVE flp1_quill_install_boot

Now the original INSTALL_BAS program can be used to change the PRINTER_DAT file by entering: "LRUN flp1_quill_install_boot" but don't forget that the DEV_USE settings will still be in force when the program finishes.

DEV in more detail: The DEV driver is also usable from SuperBasic. However, this is not something we recommend. It is easy to get confused about the settings; if in doubt don't use DEV.

DEV_USE 1, ram1_	dev1_ equivalent to ram1_
DEV_USE 2, flp1_letters_	dev2_ is flp1_letters_
DEV_USE 3, win1_work_new	dev3_ is win1_work_new

NOTE: unlike PROG_USE and DATA_USE, the underscore at the end is significant. Thus, after entering the above commands

OPEN #3, dev1_f1	Opens ram1_f1
OPEN #3, dev2_bankmanager	Opens flp1_letters_bankmanager
OPEN #3, dev3_f1	Opens win1_work_newf1
DELETE dev3_junk	Deletes win1_work_new_junk

There is a variation on the DEV_USE call which enables the setting up of default chains. If you put another number at the end of the DEV_USE command it will be taken as the DEV to try if the open fails. This next DEV can also chain to another DEV. The DEV driver stops chaining when all DEVs in the chain have been tried.

DEV_USE 1,ram1_,2	dev1_ is equivalent to ram1_
DEV_USE 2,flp1_latest_,3	dev2_ is flp1_latest
DEV_USE 3,win1_work_,1	dev3_ is equivalent to win1_work_

LOAD dev1_Prog_bas	Tries ram1_Prog then flp1_latest_Prog_bas then finally win1_work_Prog_bas
LOAD dev2_DiskCheck	Tries flp1_latest_DiskCheck_bas then win1_work_DiskCheck and finally ram1_DiskCheck

DELETE does not chain with DEV.

Examining DEV settings: The command DEV_LIST and two functions DEV_USE$ and DEV_NEXT$ can be used to examine the DEV allocations.

DEV_LIST	Lists current DEVs in #1
DEV_LIST #2	Lists current DEVs in #2

PRINT DEV_USE$(3)	Prints the usage for dev3_
PRINT DEV_NEXT(1)	Prints the next DEV in the chain after dev1_

Interaction between DATA_USE, PROG_USE and DEV: If you are going to use the DEV defaults, it makes sense to set the DATA_USE and PROG_USE defaults to use DEV, and when moving from directory to change the DEV definition rather than the DATA_USE.

DATA_USE dev1_	Current directory is dev1_
DEV_USE 1,flp2_myprogs_	which is myprogs on drive 2
PROG_USE dev2_	Programs from dev2_
DEV_USE 2,flp1_ex_,1	which is flp1_ex_ or flp2_myprogs_

Changing the DEV name: The DEV name can be changed by specifying a three letter name or string. DEV_USE with no parameter resets the name to DEV.

DEV_USE 1,flp2_myprogs_	dev1_ is myprogs_ on drive 2
DEV_USE 2,flp1_ex_,1	dev2_ is flp1_ex_ or flp2_myprogs_
DEV_USE flp	flp1_is now really flp2_myprogs_and flp2_is flp1_ex_etc.
DEV_USE	flp1_ is now flp1_again

Using DEV_USE in this way is at best confusing. Take time to experiment with DEVs. Only incorporate it once you fully understand its purpose and operation.

DEV operates by intercepting the QDOS open call and redirecting the open to the appropriate device driver. It produces very little overhead on an open call and thereafter the real device driver is accessed directly by QDOS. There are no spurious channels opened. Unlike some defaulting schemes, it does not prohibit access to directories which are not the default. Thus it is usable without affecting the computer's normal operation.

RAM SIZE

Some software fails if the QL has too much memory. The QL's memory can be reduced with the RES_SIZE command.

RES_SIZE 14*64

will cause the QL to reset to give a capacity of 896K making the system look as though there is a TRUMP CARD 768K installed. A single reset sequence occurs after this command. RES_SIZE 128 is identical in operation to RES_128. Resetting to 128K has two other effects; only double density disks can be accessed and PROT_DATE 1 is executed, explained below. Only allocate memory as RES_SIZE n*64 where n is an integer from 2 to 30.

Battery Backed Clock: When the GOLD CARD is first installed the clock needs setting. This must be done using SDATE, not ADATE, which ensures the clock is configured correctly. For example:

SDATE 1992,2,24,15,30,0

will set the clock to 24 Feb 1992, 15:30. The time is maintained by the GOLD CARD after the power is switched off. Removal of the GOLD CARD from the QL can cause the time to be lost. All SuperBasic commands and QDOS calls relating to time and date affect the GOLD CARD clock. The GOLD CARD's clock can be protected using the new command, PROT_DATE, e.g.

PROT_DATE 1

In this protected state all operations using the clock only use the QL's clock. On power up or reset, except after RES_SIZE 128 or RES_128, the equivalent of

PROT_DATE 0

is executed which allows the GOLD CARD clock to be modified. The QL clock is only set to the GOLD CARD time at reset. The battery in the GOLD CARD should keep the time for about 5 years.

QL HARD DISK EXTENSIONS (for QL Hard Disk users)

All the QL HARD DISK extensions contained in the "WIN_REXT" file except WIN_EXT have been included in the GOLD CARD ROM so the file "WIN1_WIN_REXT" must not be loaded when a GOLD CARD is fitted. Instead of loading "WIN_REXT" use TK2_EXT.

Slowing the Gold Card Down: Some programs, mostly games, are rendered unusable by the speed of the GOLD CARD, even after RES_128.

SLUG 15

will delay all subsequent reads of the keyboard by 15 thousandths of a second (milliseconds) which is a useful starting value. As you get better at the game, reduce the delay for a more challenging play!

SLUG works by delaying the QDOS keyboard read routine. As a consequence the program will speed up when not reading the keyboard. However,the program reads the keyboard when interacting with the user so this effect shouldn't be a problem.

We know of only one game that duplicates the keyboard read routine so is not slowed by SLUG. Nothing can be done, short of a major hardware change, to slow programs of this type.

Extra FILING SYSTEM FACILITIES on the Gold Card

Reading the dates and the version number: Whenever a file which has been modified is closed the file is marked with the date and time when it was closed. This is called the update date. It is calculated in seconds from the beginning of 1961. The extended filing system also maintains a file version number which is incremented when a modified file is closed. There is also a facility to set a 'backup' date in a file header to record the most recent backup copy of the file.

There are two new functions to compliment the Toolkit II function FUPDT. All three return a floating point value. They can be used to find the update date, the backup date and the version number of a file which has already been opened. If the channel number is not specified it will default to #3. Also they can be used to open a file, find the date or version and close the file. In this case, the filename should be specified but preceded with the \ character. The filename can be given as a string or a name and will default to use the data default directory. Examples are:

```
PRINT FUPDT                    Print update date of file on #3
value=FUPDT (#5)               Get file update date for file already open on #5
value=FUPDT (\fred)
        or
value=FUPDT (\"fred")          Get file update date for fred in the data default
                               directory

value=FBKDT (#5)               Get file backup date for file already open on #5
value=FBKDT (\"fred")          Get file backup date for fred
value=FVERS (#5)               Get version number for file already open on #5
value=FVERS (\"fred")          Get version number for fred
```

Setting the dates and version number: There are three procedures to set the update date, the backup date and the version number of a file. Like the three functions for reading the dates and version number, they can be used with either a channel number (default #3) or a filename, preceded by a \. The file name can be a name or a string and uses the data default directory.

A date or version number of 0 will have the same effect as omitting it. A date or version number of -1 will have no effect on the file. If the update date has been set it will not be reset when the file is closed. If the version number has been set it will not be incremented when the file is closed. Examples of use of SET_FUPDT, SET_FBKDT and SET_VERS:

```
SET_FUPDT #5                          Set update date to now
SET_FUPDT \flp1_fred,DATE-24*60*60    Set update date of 'fred'
                                      on flp1_ to 24 hours ago

SET_FUPDT \flp1_fred                  Set existing update date to now
```

SET_FBKDT # channel	Set backup date to now
SET_FBKDT \filename	
SET_FBKDT #channel,date	Set backup date to date
SET_FBKDT \filename,date	

Version numbers can be manipulated using:

SET_FVERS #5	Do not increment version number of file open on #5
SET_FVERS #5,1	Set version number of file on #5 to 1
SET_FVERS \flp1_fred,2	Force version number to 2

DIRECT SECTOR READ/WRITE: The disk controller on the GOLD CARD supports high density (HD) and extra high density (ED) disk drives. Consequently, the software has been extended to support direct sector read/write with these disk densities.

To directly access the sectors on an HD diskette use the file name

FLPn_*D2h where n is the drive number

ED diskettes formatted under QDOS use the file name

FLPn_*D4e

The 4 indicates that the sectors are 2048 bytes long.

If direct sector read/write is performed from SuperBasic the file names above must be enclosed in quotes. E.g. with Toolkit II enabled and an ED diskette in FLP2_

```
OPEN #3,"flp2_*d4e"
GET #3 \ 1+0*256+0*2^16, Sector$
```

will read the first sector of side 0, track 0 on the diskette into the variable Sector$. Note that INPUT #3, Sector$ is unsuitable as there may be CHR$(10)s anywhere in the sector causing some of the data to be missed. The first four characters of Sector$ should be QL5B. The next ten characters are the name of the diskette as shown with DIR FLP2_.

Following the two program lines above with the three below illustrates how to change the name of a diskette.

```
Sector$(5 TO 14)="CustomName" : REM The desired name
PUT #3 \ 1+0*256+0*2^16, Sector$
CLOSE #3
```

For more about direct sector read/write see the revelent section of the TOOLKIT II manual that follows this supplement.

Miracle Systems Ltd

TOOLKIT II
MANUAL

20 Mow Barton, Yate, Bristol, BS17 5NF, UK
Telephone/Fax: (01454) 883602

BEGINNERS GUIDE

The extra commands offered by Toolkit II are enabled by entering TK2_EXT after pressing F1 or F2 after switch-on.

A printer buffer for the serial ports is created by entering PRT_USE SER,SER.

To share one printer between 2 or more networked QLs firstly enable Toolkit II by entering TK2_EXT, set up the QLs station numbers by entering NET 1 on 1 QL, NET 2 on the next and so on, enter FSERVE on each QL, then on each remote QL use the device name N1_SER1 assuming the printer is attached to the first QL. For example, after entering FSERVE, put QUILL in MDV1 on QL 2 and enter LRUN MDV1_BOOT. To print, instead of using the default "printer" option, type in _N1_SER1. The document will be printed via SER1 on QL 1.

For disk sharing the QLs must be set up in a similar manner to printer sharing. If the disk is attached to QL 3 then the file "fred" would be loaded into QL 2 by entering LOAD N3_FLP1_FRED on QL 2.

To run programs unable to run on expanded QLs like Psion Chess enter RES_128 after F1 or F2 after switch on. The QL will now look as though it is unexpanded to the software.

To use QUILL from DISK instead of MICRODRIVE: If you only have a single disk drive, then use the CONFIG_BAS routine on your QUILL cartridge so that QUILL will expect SYSTEM, DATA and HELP information from MDV1_.

1. Reset the QL.
2. Press F1 or F2.
3. Type TK2_EXT then press ENTER.
4. Put a blank disk into FLP1_.
5. Type FORMAT FLP1_QUILL then press ENTER.
6. When the disk has formatted put your QUILL cartridge into MDV1_.
7. Type WCOPY MDV1_ TO FLP1_ then press ENTER.
8. A message saying MDV1_nnn TO FLP1_nnn..Y/N/A/Q? will appear. (Where nnn can be any name eg. BOOT, CLONE, etc.)

9. Press the letter A to copy all the files.
10. When the copying has finished type LOAD FLP1_BOOT then press ENTER.
11. Once the file has loaded type RENUM then press ENTER.
12. Type 90 FLP_USE MDV then press ENTER.
13. Type SAVE FLP1_BOOT then press ENTER.
14. A message saying FLP1_BOOT EXISTS, OK TO OVERWRITE..Y OR N? will appear.

15. Press the letter Y to overwrite the file.

16. When the light on the disk drive goes out, the disk is ready to use.
17. After RESET, once the disk has loaded in, any reference to MDV will access the disk instead of microdrive.

18. Use the same procedure to copy your other PSION programs to disk.
19. You can also use the WCOPY command to copy data from microdrive to disk.

DISCLAIMER: In no circumstances will Miracle Systems, 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 GOLD CARD family's software.

The Floppy Disk Driver and RAM disk:

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 emulation command "FLP_USE MDV" to all BOOT files, or type this command at the start of a session on the QL. The effect of this command is to make the floppy disks pretend to be rather large and fast Microdrives.

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

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

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

FORMAT flp1_test	formats a new floppy disk in drive 1
DIR flp1_	directory listing of floppy disk 1
SAVE flp1_myprog	save the current SuperBASIC program as "myprog" in floppy disk 1

OPEN_NEW #3, flp2_data	creates and opens a new file "data" in floppy disk 2
COPY mdv1_x TO ram1_x	copies file x from Microdrive 1 to floppy disk 1

The term "RAM disk" is a misnomer. It is used to denote a "virtual" device (one that 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	creates a new RAM disk 2, see below
DIR ram1_	directory listing of RAM disk 1
SAVE ram1_myprog	save the current SuperBASIC program as "myprog" in RAM disk 1

OPEN_NEW #3, flp2_data	creates and opens a new file "data" in RAM disk 2
COPY mdv1_x TO ram1_x	copies file x from MDV 1 to RAM disk 1

RAM Disk Creation: A dynamic RAM Disk is created just by accessing it with any normal operation (e.g. DIR). This type of RAM Disk takes memory as required, and releases any memory as files are deleted or truncated. A fixed RAM disk is created by formatting it: the size, in sectors, is given in place of the usual medium name. This pre-allocates all the space that will be available in the RAM disk.

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 8, inclusive, while the number of sectors (512 bytes) is only limited by the memory available. A RAM disk can be formatted from the FILES menu of QRAM.

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 fixed RAM disk to be a permanent feature until the QL is reset.

Using a fixed RAM Disk not only reduces the danger of heap fragmentation, but also provides higher access speeds during file creation. Since it always occupies the maximum space you ever wish to use, it is much less flexible.

Microdrive Imaging: Microdrive imaging is a very fast method of loading files from a Microdrive cartridge. To produce a Microdrive image, a RAM Disk is formatted with the name of the Microdrive required:

FORMAT ram1_mdv2 loads an image of mdv2 into RAM Disk 1

The RAM Disk can even load a Microdrive with a damaged directory. It cannot, however, load a Microdrive with a damaged map. The RAM Disk will try up to 3 times to read a faulty sector. If it fails, the number of good sectors returned from the format will be fewer than the total number. Any file with bad sectors will be marked with an"*" in the RAM Disk directory.

Floppy and RAM Disk Compatibility

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

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

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

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

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

TRUNCATE Trap #3, D0=4B

This call truncates a file to the current byte position.

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

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

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

Microdrive Emulation: The standard drivers also include the SuperBASIC procedures FLP_USE and RAM_USE to change the name of the floppy disk and RAM disk drivers.

FLP_USE mdv or FLP_USE 'mdv'

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

RAM_USE mdv or RAM_USE 'mdv'

will do the same for the RAM disks. For Example

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

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

Any three letters may be used as a new device name, in particular

FLP_USE flp and RAM_USE ram

will reset the drivers to their 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 cost of reduced immunity to erroneous disk swapping. There are three levels of security, the lowest level still being at least as secure as common disk based operating systems (e.g. MSDOS and CPM).

A user may specify the time taken for the disk drive motor to get the disk speed to within the specification. A user may specify the number of tracks to be formatted on a disk. The parameters are specified by three commands:

FLP_SEC	security level
FLP_START	start up time (in 50ths of a second)
FLP_TRACK	number of tracks

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

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

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

Security Level 0: The disk is only checked when a file is opened and the drive has stopped since the last time it was checked and there are no files already open on the drive. The map is only updated after a file is closed (or flushed) when half a second has elapsed without any other disk operation.

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

Security Level 1: The disk is checked when a file is opened, or data or the map is to be written, and the drive has stopped since the last time it was checked. The map is only updated after a file is closed (or flushed) when half a second has elapsed since the previous disk operation.

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

Security Level 2: The disk is checked whenever a file is opened or whenever the map or data is to be read from or written to the disk and the drive has stopped since the last time the disk was checked.

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

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

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

disk name read/write failed

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

disk name files still open

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Disk format ->	40T SS	40T DS	80T SS	80T DS
Drive				
40T SS	OK	?	X	X
40T DS	OK	OK	X	X
80T SS	RO	?	OK	?
80T DS	RO	RO	OK	OK

OK = compatible RO = read only ? = incompatible but may not be detected on some drives.

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

```
FORMAT 'FLP1_DISK_NAME *'
```

Dynamic Printer Buffer: The Printer Buffer has two names: the "usage" and the "device". The default usage name is PRT and the default device is SER. The printer buffer works by intercepting any OPEN call to a device whose name starts with its usage name. It substitutes the device name for the usage, and tries to open the device. If it succeeds, then all the output is buffered within the QLs main memory. If the device is in use, then the output is also buffered until the device is available. There is no limit to the number of buffered output files open at one time. If an error occurs during output, the buffer contents are thrown away. The current printed output may also be thrown away by the command:

```
PRT_ABT
```

This will ABORT the file with the message "****** ABORTED ******"

Using the default usage and device all references to a device called PRT will use the serial driver SER. Any parameters appended to the PRT name will be transferred to the SER name:

OPEN #3, prt	will open SER with a buffer
COPY flp1_fred, prt2c	will copy to SER2C with a buffer

The usage and device can be changed by using the command

PRT_USE usage, device

Two cases are particularly useful. In the first, the usage and device names are the same. This has the effect of introducing a buffer transparently into a device. In the second, the device name is of zero length. This means that the usage name may be followed by any device name.

PRT_USE ser,ser	buffer all output to SER1 and SER2
PRT_USE b_,''	b_ser1 is buffered SER1, b_par is buffered PAR etc.

Screen Dumps: The screen dump facilities are available in three ways. Screen dumps may be invoked at any time with a user definable hotkey, screen dumps may be invoked using a SuperBASIC command or screen dumps may be made through the IO system from programs written in any language.

SuperBASIC Commands: There are four SuperBASIC commands for screen dumps. The principal command is SDUMP. This has three formats, which are self explanatory:

SDUMP	dump whole screen
SDUMP #channel	dump SuperBASIC window
SDUMP width, height, x origin, y origin	dump specified pixel area
SDUMP address of flag	dump save area
SDUMP address, width, height, x origin, y origin	dump window within save area

The last two forms are to dump all or part of a window save area as saved by the QJUMP pointer interface command PSAVE.

The other three commands control the dump facilities. SDP_KEY sets the single key dump facility. When activated, pressing ALT and the specified key will invoke the "hotkey" dump routines.

SDP_KEY p	select key "p" for hotkey dump
SDP_KEY 'p'	... the same
SDP_KEY	inhibit hotkey dump

SDP_DEV sets the screen dump device. The default is "SER", this may be changed to any QDOS device or file name.

SDP_DEV 'ser2'	dump to SER2
SDP_DEV n2_par	dump to PAR on network node 2
SDP_DEV flp2_dump1	dump to DUMP1 on FLP2

SDP_SET sets the printer type, scaling, and print method. You may give between one and four parameters for this command.

SDP_SET printer number, scale, inverse, random

The printer numbers and scales are given in the following table. The "inverse" and "random" parameters are either true or false (non-zero or zero). An "inverse" dump prints black for white and vice versa. A random dump provides some impression of grey scales even at one dot per pixel. If a parameter is not given, the setting remains unchanged.

SDP_SET 2,3,0	select Epson FX80, scale 3, not inverse (random unchanged)
SDP_SET 1,1,1,0	select Epson MX80, scale 1, inverse, not random (this is the default)

IO Device SDUMP: The SuperBASIC commands all access the screen dump software through the IO device SDUMP.

The screen dump parameters may be set by sending bytes to the device SDUMP using the IO.SBYTE or IO.SSTRG IO calls. A dump is invoked by defining a window using the SD.WDEF call. In the following examples, SuperBASIC is used for simplicity. Similar calls may be made from any language.

To set the various parameters of the screen dump routines, a code byte is sent to the device followed by the parameter.

Code	Parameter	
0	character	sets the screen dump hotkey
0	0	suppresses the screen dump hotkey
1	byte	sets printer number
2	byte	sets scale
3	byte	sets inverse flag
4	byte	sets random flag
8	standard string	sets device name
9	byte length string	sets device name

Note that the streams of bytes 8,0,3,65,66,67 and 9,3,65,66,67 both set the dump device name to "ABC". The bytes may be sent one at a time (IO.SBYTE) or many at a time, using the send multiple bytes (IO.SSTRG) operation.

To dump an area of the screen, a call is made to set the "window" area using SD.WDEF. To dump from the screen, the border width (D2) should be specified as zero.

```
OPEN #4,'sdump'              open dump device
BPUT #4,1,2,2,3              set printer 2, scale 3
BPUT #4,8                    set dump device ...
PUT #4,'n2_ser'              ... to "N2_SER"
WINDOW #4,256,202,256,0      dump window (mode 4 SuperBASIC #1)
CLOSE #4                     done
```

The parameters which have been set remain set until reset by further calls. Closing the SDUMP channel has no effect other than to keep the QL tidy.

The parameter setting calls will always complete immediately, the window definition call will complete when the dump is finished. If zero timeout is specified, then the dump will continue after the window definition call has returned not complete.

From assembly language, D2 is a key defining the area of memory to be dumped. If D2 is 0, the dump will be from the screen memory. If D2 is set to 1, then the partial save area pointed to by A2 will be dumped. If D2 is 2, then a window (defined by (A1)) within the partial save area will be dumped. If D2 is non-zero, A2 must be set.

Screen Dump Formats

Printer	scale	dots /in	lines /in	dots 512	max ratio width		dots 256	max ratio width	
1 Epson MX80	1	120	72	1x1	512	1.23			
or similar	1	60	72				1x1	256	1.23
	2	60	72	1x2	480	1.23	2x2	240	1.23
	3	120	72	2x2	480	1.23	4x2	240	1.23
2 Epson FX80	1	90	72	1x1	512	0.92			
additional	1	60	72				1x1	256	1.23
formats	2	90	72	1x1	512	0.92	2x1	256	0.92
	3	90	72	2x2	360	0.92	4x2	180	0.92
3 Epson FX100	1	90	72	1x1	512	0.92			
wide	1	60	72				1x1	256	1.23
carriage	2	90	72	1x1	512	0.92	2x1	256	0.92
	3	90	72	2x2	512	0.92	4x2	256	0.92
4 Epson JX80	1	90	72	1x1	512	0.92			
	1	60	72				1x1	256	1.23
	2	90	72	1x1	512	0.92	2x1	256	0.92
	3	90	72	2x2	360	0.92	4x2	180	0.92

Screen Dump Formats - (cont'd)

5 Epson	1	80	60	1x1	512	0.99			
LQ2500	1	60	60				1x1	256	1.48
8 pin	2	120	60	2x1	512	0.74			
	2	80	60				2x1	256	0.99
	3	80	60	2x2	512	0.99	4x2	256	0.99
6 Epson	1	120	180	1x2	512	0.99	1x1	256	0.99
LQ2500	2	180	180	2x3	512	1.11	3x2	256	0.99
24 pin	3	180	180	3x4	512	0.99	6x4	256	0.99
7 Epson	1	80	60	1x1	512	0.99			
LQ2500	1	60	60				1x1	256	1.48
8 pin	2	120	60	2x1	512	0.74			
colour	2	80	60				2x1	256	0.99
	3	80	60	2x2	512	0.99	4x2	256	0.99
8 Epson	1	120	180	1x2	512	0.99	1x1	256	0.99
LQ2500	2	180	180	2x3	512	1.11	3x2	256	0.99
colour	3	180	180	3x4	512	0.99	6x4	256	0.99
9 Brother HR4	1	120	72	1x1	512	1.23			
	1	60	72				1x1	256	1.23
	2	60	72	1x2	480	1.23	2x2	240	1.23
	3	120	72	2x2	480	1.23	4x2	240	1.23
10 Olivetti	1	110	72	1x1	512	1.13			
JP101	1	110	108				1x1	256	0.75
	2	110	108	1x1	512	0.75	3x2	256	1.00
	3	110	72	2x2	440	1.13	4x2	220	1.13
11 Seikosha	1	60	63	1x1	480	0.70	1x1	256	1.41
GP-100A	2,3	60	63	1x2	480	1.41	2x2	240	1.41
12 Seikosha	1	60	72	1x1	480	0.61	1x1	256	1.23
GP_250X	2,3	60	72	1x2	480	1.23	2x2	240	1.23
13 Seikosha	1	80	80	1x1	512	0.74	1x1	256	1.48
GP-700A	2	80	80	1x2	512	1.48	2x2	256	1.48
	3	80	80	1x2	512	1.48	3x2	212	0.99
14 Canon	1	80	80	1x1	512	0.74	1x1	256	1.48
PJ1080A	2	80	80	1x2	512	1.48	2x2	256	1.48
	3	80	80	1x2	512	1.48	3x2	212	0.99
15 Centronics	1	75	72	1x1	512	0.77	1x1	256	1.42
739	2	75	72	1x1	512	0.77	2x1	256	0.77
	3	75	72	2x2	300	0.77	3x2	200	1.03

Screen Dump Formats - (cont'd)

16 C.Itoh	1	120	72	1x1	512	1.23			
7500	1	160	72				1x1	256	1.23
	2	160	72	2x1	512	0.82			
	2	120	72				2x1	256	1.23
	3	120	72	2x2	480	1.23	4x2	240	1.23
17 Toshiba	1	180	180	1x2	512	1.48	2x2	256	1.48
TH 2100H	2	180	180	2x3	512	1.11	3x2	256	0.99
24 pin	3	180	180	3x4	512	0.99	6x4	256	0.99
18 Brother	1	70	72	1x1	512	0.72	1x1	256	1.44
8056	2	70	72	1x1	512	0.72	2x1	256	0.72
	3	70	72	2x2	280	0.72	3x2	186	0.96
19 EpsonMX100	1	120	72	1x1	512	1.23			
or similar	1	60	72				1x1	256	1.23
	2	60	72	1x2	512	1.23	2x2	256	1.23
	3	120	72	2x2	512	1.23	4x2	256	1.23
20 Tandy	1	100	72	1x1	512	1.03			
DMP 105	1	60	72				1x1	256	1.23
	2	60	72	1x2	512	1.23			
	2	100	72				2x1	256	1.03
	3	100	72	2x2	400	1.03	4x2	200	1.03
21 OKI	1	100	66	1x1	512	1.12			
Microline	1	60	66				1x1	256	1.35
82 / 84	2	100	66	1x1	512	1.12	2x1	256	1.12
OK Writer	3	100	66	2x2	400	1.12	4x2	200	1.12
22 Fastext 80	1	72	72	1x1	512	0.74			
	1	60	72				1x1	256	1.23
	2	60	72	1x2	480	1.23	2x2	240	1.23
	3	72	72	2x3	288	1.11	3x2	192	0.99
23 MT-80	1	85	82	1x1	512	0.77	1x1	256	1.53
	2	170	82	2x1	512	0.77	3x1	256	1.02
	3	170	82	3x2	425	1.02	6x2	212	1.02

Preface

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

QL Toolkit II is, therefore, a revised (to the extent of being almost completely rewritten) and much enlarged version of the original QL Toolkit. 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 QL Toolkit 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 Stuven of QSOFT, Denmark, whose indomitable faith in the technical merit of this product has kept me on my toes. My thanks to you all, TT.

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 hostile 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
bold letters	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! Brother, I give up). 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 only allows 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 same name as 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 and 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). Others 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 '\' 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\files	list current directory to file 'files'

this last example should be distinguished from

DIR files list directory entries starting with files 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 around 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: SuperBASIS 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 developement 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

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 director 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	lists 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
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,	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 specifications	

Section 9 Job Control: The mulitasking 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
JOB$ (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

FTEST (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
FXTRA (#channel)	find file extra info
FNAME$ (#channel)	find filename
FUPDT (#channel)	find file update date

Section 12 Direct Access Files:

Toolkit II 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 on to 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, list of items to print	fixed format output

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_DEFB	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 name.

Section 17 Error Handling: These facilities are provided for error processing in versions JS and MG of SuperBASIC.

ERR_DF	true if drive full error has occured
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
ALARM 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 Devices

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

Section 20 Console Driver: Toolkit II provides last line recall for the command channel #0 as well as allowing strings of characters to be assigned to 'ALT' key strokes received on this channel.

Commands

<ALT><ENTER>	keystroke recovers last line typed
ALTKEY character, string	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 respecified 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 disk system or printer.

Commands

FSERVE	invokes the 'file server'
NFS_USE name,network names	sets the network fileserver name

Device names

Nstation number_IO device	the name of a remote IO device (e.g. N2_FLP1_ is floppy 1on network station 2)

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 #0 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, 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 moved 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 between the two modes use 'SHIFT F4' (press SHIFT and while holding it down 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 arrorw	move cursor down a line
ALT up arrow	scroll up a line (the screen moves down!)
ALT down arrorw	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 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:

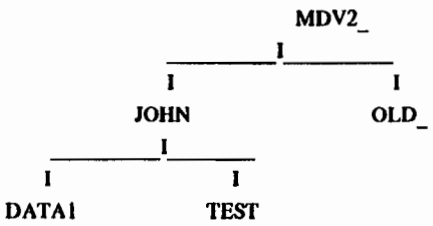
VIEW name	View file 'name' in window #1
VIEW #channel, name	View file 'name' in given window
VIEW\name1, name2	Send file 'name2' to 'name1'

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

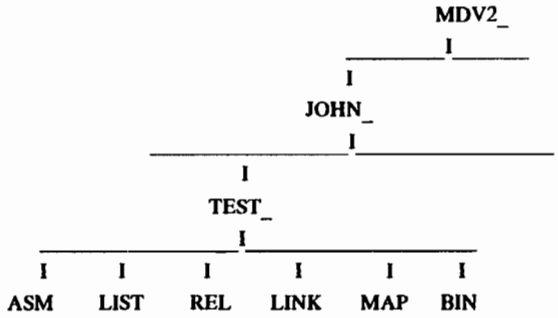
In QDOS terminology, a 'directory' is where the system expects to find a file. This can be as simple as the name of a device (e.g. MDV2_ the name of the Microdrive number 2) or be much more complex forming part of a 'directory tree' (directories grow on trees - honestly, they do). For example: the directory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



This shows another characteristic of the 'directory tree': it grows downwards. The complete QDOS filename for DATA1 in this example is MDV1_JOHN_DATA1. (You may have come across the terms 'pathname' or 'treename': 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.

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 Toolkit 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 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 procedures 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, while 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 the DEST_USE command which does not append '_' to the name given. Notionally 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 flp2_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

```
SPL_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 '_' (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:

	data	program	destination
initial values	mdv2_	mdv1_	ser
DDOWN john	mdv2_john_	mdv1_	ser
DNEXT fred	mdv2_fred_	mdv1_	ser
PROG_USE mdv2_fred	mdv2_fred_	mdv2_fred_	ser

DNEXT john	mdv2_john	mdv2_john_	ser
DUP	mdv2_	mdv2_	ser
DEST_USE mdv1	mdv2_	mdv2_	mdv1_
DDOWN john	mdv2_john_	mdv2_john_	mdv1_john_
SPL_USE ser1c	mdv2_john_	mdv2_john_	ser1c

4.4 Taking Bearings

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

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

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$))='_':      PRINT 'Destination'! DEST$
```

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.

5 File Maintenance:

The standard file maintenance procedures 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_freda_list
flp1_old__list	flp1_old__list	flp1_old_jo_list flp1_old_freda_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 #
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 a 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 general 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_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)

These commands can be given with one or two names. The separator 'TO' is used for clarity, 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 mdv2_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 seperator 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. Otherwise confirmation will be sent 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 data 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 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_	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>) can be sent to the printer at the end of a file.

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

The separator TO is used for clarity, you may use a comma instead. The normal use of this command is with one name only:

SPL_USE ser	set spooler default
.....	
SPLF fred	spool fred to ser, 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 #channel3 TO #channel2

where channel 3 must have been opened for input and channel2 must have been opened 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 Program: 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 a file.

DO name	do commands in file

The commands should be 'direct': 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 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

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 rewritten to avoid the problems that occur in AH and JM ROMs when a CALL is made from a large (>32kbytes) 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 O 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 for 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 it 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 #channels; 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 the data are termed 'filters'.

Using the symbols [ ] to represent a single optional item
( ) to represent a repeated optional item

the complete form of the EX command is

EX [#channel TO] 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 I/O 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 lno 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 PAGE. 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.

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 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 Toolkit II) 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.

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' if 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
SPJOB 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 DO 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
JOB$ (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 O, otherwise it will return the id of the next Job in the tree.

Job O always exists and owns directly or indirectly all other Jobs in the QL. Thus a scan starting with id = O and top Job id = O 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 JOB$= ' ' NXJOB=-1

10 Open and Close: All of the 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 #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

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$) < > -7: PRINT 'File '; file$; '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:

```
ouch = FOP_NEW (fred) :                REMark open fred
if ouch < 0: REPORT ouch:STOP:         REMark ... oops
PRINT #ouch, 'This is file Fred'
CLOSE #ouch
```

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 a 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 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	0 for ordinary files
	1 for executable program
	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, 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).

12.1 Byte I/O

BGET #channel\position, item	get bytes from a file
BPUT #channel\position, item	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 O 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 #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 I/O - It is possible to put or get values in their internal form. The PRINT and 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 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 #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).

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.

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 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.

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 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 ', then all the following characters are written out until the next " or '.

If the \ character is found, then a newline is written out.

All the other special characters appear in format '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)
-#.####!!!!	exponent form (e.g. 2.9979E+08) optional sign
+#.####!!!!	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 - 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 the 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 decimal point, the other is used as the thousands separator. 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 the $ 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$='@$ Charges  ******.** : ($SKr##.###,##) : ##,###.##+\'

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

will print

```
$ Charges ****123.45:        SKr123,45 :        123.45+
$ Charges * -12345.67:       SKr12.345,67:      12,345.67 -1.235E+06
```

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
CDEC$ (value, field, ndp)	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 of 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)	hexidecimal 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-recognized character.

14 Display Control

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

14.1 Cursor Control

- The function 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 IO commands, INKEY$ defaults to channel #0. For example:

```
CURSEN: in$=INKEY$ (#1,250): CURDIS
```

will enable the cursor in window #1, and wait for up to 5 seconds for a character from the keyboard. If nothing is typed within the 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 values 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 appropiate 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

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.

15 Memory Management

As QDOS is a multitasking operating system, there may be several jobs running in a QL, and so the amount of free memory 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)

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 definition blocks from common heap

Although there are precautions within the procedure DEL_DEFB to minimise 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	0 null	the usage is 0 unset
	1 string	1 variable
	2 floating point	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 the 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.

PARSTR$ (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 PARSTR$(n1,1), PARSTR$(n2,2), PARSTR$(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 occured
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, it 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
RETRY line number	specified line

18 Timekeeping

18.1 Resident Digital Clock

CLOCK	default clock in its 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 using the QL's system 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.

To set the clock the SuperBASIC command SDATE is used:

SDATE year,month,day,hour,minute,seconds

Example:

```
SDATE 1989,6,1,14,45,30
MODE 8
OPEN #6,'scr_156x10a32x16'
INK #6,0: PAPER #6,4
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 disk extensions, EXP_EXT expansion unit extensions) will reassert the Toolkit II definitions.

20 Console Driver

20.1 Keyboard Extensions

There are two 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. Thus a null string may be put at the end to add a newline to the string.

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

For example after the command

```
ALTKEY 'r', 'RJOB "SPL"  ',' '
or  ALTKEY 'r', 'RJOB "SPL"  ' & CHR$(10)
```

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 Microdrive 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 is 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 are:

NET0_station number	output to station number
NET0_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 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 a broadcast into 10 kbyte 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 IO resources attached to one QL to be accessed from another QL. This means that, for example, disk 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 IO resource. This version of the file server is more general than the first version in that the IO 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 disk drives).

FSERVE	invokes the 'file server'

There may be more than one QL on a network with a file server running: the station numbers 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, is 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 file servers 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 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_120x20a0x0	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 n1_win1_progs	by default all programs will be loaded fromm 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 a network file server.

NFS_USE name, network names	sets the network file server name

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 a 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

FLP_USE FLP is invoked after reset so if FLP is to be used as the device name in the NFS_USE command remember to include FLP_USE XXX. This will stop the QL from trying to access its own disk port instead of the network.

FLP_USE xyz	set device name for floppies to xyz
NFS_USE flp,n1_flp1_,n1_flp2_	any reference to 'flp1' on this QL will access flp1 on net station 1, etc.

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 particularily pretty messages are required, then the graphics facilities of SuperBASIC may also be used. The only standard IO facilities not available across the network are SD.EXTOP (extended operations) and SD.FOUNT (setting the founts). For example

```
ch = FOPEN (n2_con_150x10a0x0): CLS #ch
INPUT #ch,'Do you want coffee? ';rep$
IF 'y' INSTR rep$ = : 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 I/O 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 of file from the primary input should, directly or otherwise, indicate to a program that the data is complete.

23.1 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 header 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 job's 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 parameter 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 - 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$,1) &  ' -errors scr'
140 END DEFine asr
150 :
160 REMark assemble with listing
170 :
180 DEFine PROCedure asl (file$)
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-no
260 END DEFine lk
```

If the data default directory is 'FLP1_JUNK_', then the procedure calls ASR '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 B

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) hackiv	set net active 22us	wait for active
5) hbytes	for each byte 11.2us start (inactive)	for each byte wait for start (inactive)
	bit, 8*11.2us data bits, 5*11.2us	bit, read 8 data bits, if fails : restart
	stop (active) bits	
6) hackw	wait for 2.5ms for active, if not active: restart	set net active 22us
7) hackbt	wait for start bit, read 8 bits, if error: restart	send 11.2us start bit 8 data bits 00000001
C) data		
8) dactiv	set net active 22us	wait for active
9) dbytes	for each byte 11.2us start	for each byte wait for start (inactive)
	(inactive) bit, 8*11.2us data bits	bit, read 8 data bits, if fails: restart
	5*11.2us stop (active) bits	
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 activity occurs: restart	
2) wait		waiting for activity (a scout) every 20ms check IPC for BREAK
3) scout	send a scout of duration <530us, if contention occurs: restart	wait for 530us
4) scext	send a scout extension of 5ms active	
b) header		
5) hbytes	for each byte 11.2us start (inactive)	for each byte wait for start (inactive)
	bit, 8*11.2us data bits, 5*11.2us stop (active) bits	bit, read 8 data bits, if fails: nack
6) hwait	leaving net active, wait 1ms	
c) data		
7) dbytes	for each byte 11.2us start (inactive)	for each byte wait for start (inactive)
	bit, 8*11.2us data bits, 5*11.2us	bit, read 8 data bits, if fails: nack stop (active) bits

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 or inactive,	wait for 2.8us of active or inactive, if inactive: restart
10) nacks	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 a standard header.

Appendix C

Toolkit II Code Sizes

	size	nr	size/nr
Base area and tables	1618	1	1618
ED	2328	1	2328
VIEW	74	1	74
Directional control (DATA_USE, DLIST etc.)	224	11	20
File maintenance (COPY, WDEL etc.)	1356	13	104
SPL, SPLF	212	2	106
BASIC (LOAD, SAVE, RUN etc.)	308	13	24
Load and save (LBYTES, SBYTES, etc.)	182	6	30
CALL	30	1	30
EX, EW	750	2(3?)	375
JOB control procedures	292	4	73
JOB information functions	102	4	25
OPEN and FOPEN	122	11	11
CLOSE	60	1	60
File header information	86	6	14
Direct access files	518	7	74
PRINT_USING	442	1	442
Decimal conversion (required for PRINT_USING)	552	4	138

Hex and binary conversions	214	4	53
Cursor control	24	2	12
Character setting (CHAR_USE, CHAR_INC)	56	2	28
Window reset (includes 48 bytes in header)	128	2	64
Heap handling	146	4	38
Heap tidy (DEL_DEFB)	62	1	62
BASIC procedure parameter type	136	4	34
ERROR handling	54	2	27
EXTRAS	68	1	68
Microdrive extensions	720	4	180
ALTKEY and last line recall	366	2	183
Network	3064	3	1021

Utility code 1674

The sizes above do not include the table entries for each BASIC extension (=name length + 3 or 4 bytes).

facilities not included in above:

RAM disk	appox	1400
Buffered printer extension	approx	500
total	approx	2400

These can be accomodated by removing about 50 of the less useful facilities.

Appendix D

Toolkit II Update Record

V2.01 First full version.

V2.02 First release version.

V2.03 Patched to prevent MG initialisation problems.

V2.04 (Jeaggi only) network eof problems fixed.

V2.05 Lost channel on OPEN_NEW (file already exists) fixed. EX EW changed so that owner is current job.

V2.06 EX EW changed for compiled programs: EX jobs owned by O, EW jobs owned by current job and now wait!

V2.07 (Sandy only) 'bad line' character wrap problem in ED fixed.

V2.08 Empty line in ED problem (introduced in V2.07) fixed Unset string parameter collapse in PRINT_USING fixed.

V2.09 PUTting randomly positioned bytes over the network should not now shuffle the contents of a file.

V2.10 RENAME with only one name does not now leave file open. The file system prompts are now sent to #0 rather than channel 0.

V2.11 Initialisation error causing loss of replacement commands (e.g. OPEN) using JM/AH ROMs and CST QDisc V1.17 and V1.18 fixed.

V2.12 Bad error message return from opening a file name that is too long changed to return "bad name". "Bad parameter" from special job opening a file specified as a string in an EX command fixed. "Not complete" from SPL fixed. Last line recall changed to reduce problems due to asynchronous modification of keyboard queue.

V2.13 Error status returned from SAVE and LIST if drive full or bad or changed medium during output. Network fixed to prevent serial IO buffer damage when interleaving serial IO with window enquiries while reading from a file.

Appendix E

Floppy disk update Record

V1.07 (not released) Write operations held pending (up to 20 sectors). Direct sector IO added.

V1.08 Microdrive interleave problem with FS.LOAD call (in V1.07 only) fixed.

V1.09 Direct sector open does not now check the drive. On seek, the track register is set to the actual track number found on the track, seek errors will not be detected, so any track may be read from any part of the disk.

V1.10 Direct sector write in FM (*DnS) does not now give read/write failed (it did work before though - just ignore the error message). This does not affect those interfaces which have MFM only.

A fatal LOAD error condition has been removed. This occurred in V1.07 onwards if:

a) a file is LOADed within .5 second of a modification to that file,
b) the file was not closed or flushed in this period
c) the directory entry for the file has become unreadable.

(There is no logical reason for conditions a and b to be met simultaneously!)

V1.11 Version 1.11 should be functionally identical to Version 1.10. The source code has been completely reorganized.

V1.12 The step rate detection procedure, which has not functioned well since version 1.09, has been fixed.

V1.13 The disk present detection routine has been changed to work reliably with index pulses as short as 10us. (A problem with extreme out-of- spec Mitsubishi 3.5" drives.)

V1.14 The FLP_OPT command or the equivalent set of commands has been added. This now gives a choice of security versus speed, and extends the range of odd drives which may be used. The disk change detection has been redesigned and the disk header handling has been improved.

The FORMAT procedure has been rewritten. It will not now detect step errors, but instead it formats and checks the disk in 5 revolutions per track (1 second, on double sided drives), or 3 revolutions per track (.6 second, on single sided drives). The check on the 11th character of a medium name (FORMAT) is not now done unless the name is at least 11 characters long. The error returns from direct sector reads have been tidied up.

The read operations used in direct sector reads now have their own read error recovery. This should improve the reliability of direct sector reads (see V1.09 above). Direct sector reads no longer clear the read buffer before attempting to read.

When checking for the presence of a disk, the driver now waits for just over one second before giving up. If there are repeated seek errors, the step rate is automatically reduced. The driver can now scatter load zero length files without getting in a knot.

V1.15 The changes in V1.15 are mainly to accomodate the 1772 control chip. Some of these may have beneficial side effects when using 1770 or 2793.

1) When first accessing a drive a check is made for 1772 step rates.
2) A compulsory 5ms settle is added after any seek: there was a problem at 2ms step rate with premature termination of a restore command.

3) The unchecked seeks at the start of the format procedure and before a direct sector read / write are now performed at a slower step rate than the normal seeks. This should reduce the chances of an undetected seek.

The sector allocation algorithm has been changed so that the first sector of a file may be allocated in track 0 when all other tracks are full. The internal messages have been moved to the base of the ROM.

Foreign language versions can now be made with simple patches. The write track procedure (for format) has been changed to improve the worst case timing margin.

V1.16 A problem with repeated checks on a changed medium, when files are still open on a previous medium, has been fixed. The FLP_EXT command clears the procedure stack. RAM disk V1.02 incorporated where appropriate.

V1.17 RAM disk V1.03 incorcorated where appropriate.
V1.18 Verify introduced on restore; additional pauses introduced on seek error recovery.
V1.19 to V1.21 identical to V1.18

Appendix F

Index and List of Differences

This index lists the SuperBASIC extensions in alphabetical order together with the usage (procedure, function or 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
ALCHP	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	14	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	
FOP_DIR	function	10	QL Toolkit	finds vacant channel
FOP_IN	function	10	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	
FTEST	function	10	new	
FTYP	function	11	QL Toolkit	
FUPDT	function	11	new	
FEXTRA	function	11	new	
GET	procedure	12	QL Toolkit	
HEX	function	13	QL Toolkit	
HEX$	function	13	QL Toolkit	
IDEC$	function	13	QL Toolkit	
JOB$	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
NSF_USE	procedure	22	new	

NXJOB	function	9	QL Toolkit	
OJOB	function	9	QL Toolkit	
OPEN	procedure	10	QL	uses default directory
OPEN_DIR	procedure	10	new	uses default directory
OPEN_IN	procedure	10	QL	uses default directory
OPEN_NEW	procedure	10	QL	uses default directory
OPEN_OVER	procedure	10	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	19	new	
TRUNCATE	procedure	12	QL Toolkit	position may be specified
VIEW	procedure	3	QL Toolkit	
WCOPY	procedure	5	new	defaults to command window uses default destination
WDEL	procedure	5	QL Toolkit	defaults to command window
WDIR	procedure	5	QL Toolkit	
WMON	procedure	14	QL Toolkit	
WREN	procedure	5	new	defaults to command window uses default destination
WSTAT	procedure	5	QL Toolkit	
WTV	procedure	14	QL Toolkit	