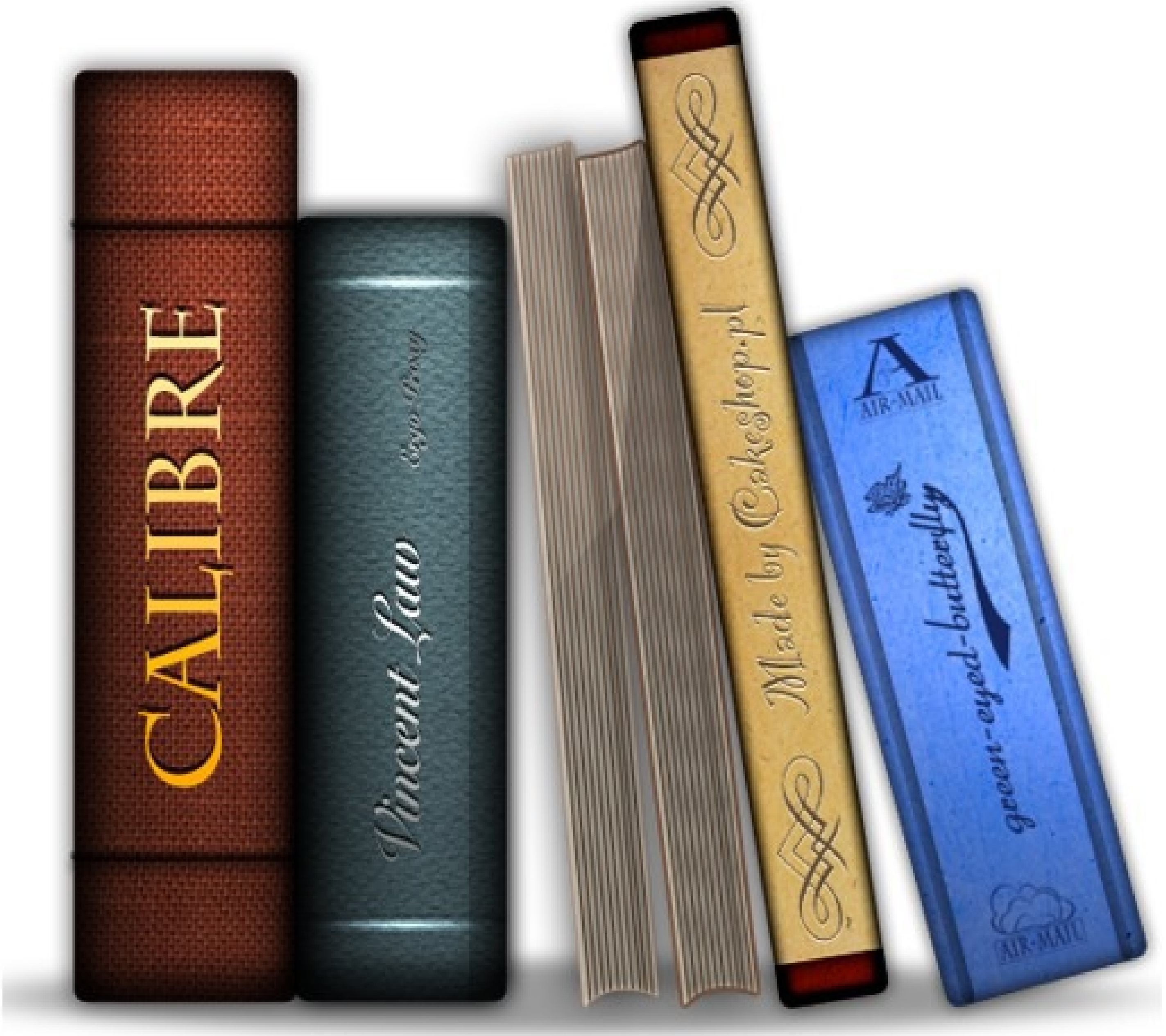
QL Addendum

Paolo Proietti

Libro 2 di Sinclair QL



calibre 0.7.35

System Variables List

The following are relative to MT_INF trap result: the $280xx can only be relied on for ROMs up to JS/MG. "+" numbers in braces are decimal offsets in the area. The base address of the system variables should be checked by using a call to the MT.INF trap (Trap #1 with D0 = 0), where the resultant value of A0 is the pointer to the system variables. Several extensions have been written for SuperBASIC to check this value, such as the function VER$(-2) in the Minerva ROM.

```
DEFine FuNction GET_SYSVARS
REMark a fairly simple system variable base address check
LET system_vars = 163840
LET v$ = VER$
IF v$ = 'JSL1' OR v$ = 'HBA' THEN LET system_vars = VER$(-2)
RETurn system_vars
END DEFine
```

$28000.W SV_IDENT Identification
$d2450000 QL (QDOS) system variable identifier
"S2AT" SMS Atari system variable identifier
Pointers defining QDOS memory map:
$28004.L SV_CHEAP Base of common heap area
$28008.L SV_CHPFR First free space in common heap area
$2800C.L SV_FREE Base of free area [+12]
$28010.L SV_BASIC Base of BASIC area [+16]
$28014.L SV_TRNSP Base of transient program area
$28018.L SV_TRNFR First free space in TPA
$2801C.L SV_RESPR Base of resident procedure area
$28020.L SV_RAMT Top of RAM(+1) [+32]
$28024.L SYS_MXFR Maximum return from free memory call (SMS only)
$28028.L SYS_RTC Real time in seconds (SMS only)
$2802C.W SYS_RTCF Real time fractional, countdown (SMS only)
$2802E.W SV_RAND Random number(constantly changing)
$28030.W SV_POLLM Count of poll interrupts missed
$28032.B SV_TVMOD 0 if not TV display [+50]
$28033.B SV_SCRST Screen status (0=active)
$28034.B SV_MCSTA Current value of MC status register
$28035.B SV_PCINT Current value of PC interrupt register
$28036.B SV_USER User number in Toolkit 3 (Note:TK3 only!)
$28037.B SV_NETNR Network station number [+55]
Pointers to the list of tasks and drivers:
$28038.L SV_I2LST Pointer to list of interrupt 2 drivers
$2803C.L SV_PLIST Pointer to list of polled tasks
$28040.L SV_SHLIST Pointer to list of scheduler tasks
$28044.L SV_DRLST Pointer to list of device drivers
$28048.L SV_DDLST Pointer to list of directory device drivers
$2804C.L SV_KEYQ Pointer to a keyboard queue
$28050.L SV_TRAPV Pointer to trap redirection table
Pointers to resource management tables:
$28054.L SV_BTPNT Pointer to most recent slave block entry. Slave
tables are of 8-byte entries, others are 4-byte.
$28058.L SV_BTBAS Pointer to base of slave block table
$2805C.L SV_BTTOP Pointer to top of slave block table
Jobs table: The jobtable is a sequence of .Ls, each pointing to
a job control block. LSW of a JobID is the position of that job's
.L in the jobtable.
$28060.W SV_JBTAG Current value of job tag
$28062.W SV_JBMAX Highest current job number
$28064.L SV_JBPNT Pointer to current job table entry
$28068.L SV_JBBAS Pointer to base of job table
$2806C.L SV_JBTOP Pointer to top of job table
Channel table:
$28070.W SV_CHTAG Current value of channel tag
$28072.W SV_CHMAX Highest current channel number
$28074.L SV_CHPNT Pointer to last channel checked
$28078.L SV_CHBAS Pointer to base of channel table
$2807C.L SV_CHTOP Pointer to top of channel table
MINERVA:
$2807C.L - pointer to new extensions, including default
fonts, cursor style, etc: see ASM.15-19
$28080.L SYS_FRBL Free Block List, to be returned to common heap
(SMS only)
Keyboard [poke only those marked with asterisk]
$28088.W SV_CAPS *Caps lock: 0=normal, -256.L=255.B=caps locked
[+136]

$2808A.W SV_ARBUF Auto repeat buffer
$2808C.W SV_ARDEL *Autorepeat delay: default=30 [+140]
$2808E.W SV_ARFRQ *Autorepeat 1/frequency: default=2 [+142]
$28090.W SV_ARCNT Autorepeat count
$28092.W SV_CQCH Keyboard change queue character code - default
ctrl-C=$03 [+146]
Misc
$28094.W SV_WP Write protect
$28096.W SV_SOUND Sound status (BEEPING)
$28098.L SV_SER1C Receive channel 1 queue address
$2809C.L SV_SER2C Receive channel 2 queue address
$280A0.B SV_TMODE ZX8032 transmit mode
$280A1.B SV.PRCS Processor type, stored in top 4 bits, hex value
of these 4 digits denotes 68000 family processor
type ($0x=68000/8, $1x=68010, $2x=68020 etc)
Bottom 4 bits contains information about any
Floating Point Unit (FPU) available. 0=no FPU,
1=internal MMU, 2=68851 MMU, 4=internal FPU,
8=68881 or 68882 FPU.
Note: not implemented on
original QLs, Gold Card onward only. Note: QPC
emulates a 68000 but identifies itself as a
68010 processor
$280A2.L SV_CSUB *If non-zero, is address of subroutine to jump
to when CAPSLOCK is pressed: the subroutine
should spoil nothing.
$280A6.W SV_TIMO Timeout for switching transmit mode (QL)
$280A6.B DMA in use (SMS, ST)
$280A7.B SV_MTYP Machine type value.
bit 0=1 Hermes (QL) or blitter (Atari) chip
installed.
bits 1 to 4 = machine type (0=ST, 2=Mega ST or
ST with RTC, 4=Stacy, 6=STE, 8=Mega STE, 10=Gold
Card, 12=Super Gold Card, 16=Falcon, 17=Q40/Q60,
24=TT030, 28=QXL, 30=QPC, 31=QLay emulator)
bit 5 to 7 = display type. 0=QL/Futura,
32=Monochrome monitor, 64=Atari Ext.4, 128=Atari
QVME, 192=QL mode LCD or VGA, 160=Aurora
$280A8.W SV_TIMOV Value of switching timeout (2 chars.)
$280AA.W SV_FSTAT Flashing cursor status
$280AC.L SV_PROGD$ Pointer to PROG_USE name (Toolkit 2 systems)
$280B0.L SV_DATAD$ Pointer to DATA_USE name (Toolkit 2 systems)
$280B4.L SV_DEST$ Pointer to DEST_USE name (Toolkit 2 systems)
PROGD$, DATAD$ and DESTD$ pointed to as word for
length followed by characters of string.
$280B8.L SV_THGL Pointer to Thing list (systems with Thing list
only)
$280E0.L - Used by QATS software (QL Applications Traffic
Supervisor)
$280E4.L - Used by QATS software
$280E8.L - Used by QATS software
$280EE.B SV_MDRUN which drive is running
$280EF.B SV_MDCNT Microdrive run-up run-down counter
$280F0.B*8 SV_MDDID Drive ID * 4 of each microdrive (8 bytes)
$280F8.B*8 SV_MDSTA status 0=no pending ops (8 bytes)
$28100.L*16 SV_FSDEF Pointers to file system physical definition
$28140.L SV_FSLST Pointer to list of file channels
$28144.B SV_XACT Translate is active flag
$28145.B - (Unknown)
$28146.L SV_XTAB Pointer to TRA table
$2814A.L SV_ERMS Pointer to message table
$28180 SYS_TOP Top of system variables, bottom of Supervisor
Stack
The following area, between $28180 and $28480 is reserved for the supervisor stack. There is no explicit stack protection in the code, although the stack should be of sufficient size for most normal purposes.

THE QL NETWORK

By David Denham

From time to time, I set up a spare QL when my grand-children come over to stay with us for a weekend. They enjoy playing games and generally messing about with a computer that's just a little bit different to the PCs and PlayStations they are used to.

Recently, one of them asked me what the little 3.5mm jack sockets on the back were for, and I explained they were for a QL network, where you could wire up several QLs together, save files to each other's computers and print to each other's computers.

Despite their tender ages, they took it all in and seemed a little surprised that a 20 year old computer could do all this, so I was coaxed into setting up a network for them to use during the weekend. As usual, I didn't really like to admit I didn't know all that much about it, so we got the manuals out and learned how to use them. They ended up teaching me, but that's another story. The network is really quite impressive despite its age and really easy to use once you get used to the principles behind it.

Before you even think about using the QL Network, make sure you use Toolkit 2. It makes it so much more of a joy to use. The basic network is useable without Toolkit 2, but you really do need to have Toolkit 2 to make the most of the network.

Although my experience of use of the QL network is purely with QLs, I'm told that Aurora and QXL cards also have QL network sockets and that they are totally or 99.99% compatible with the QL. The only factor to upset this compatibility is timing - an overclocked QXL or a Toolkit 2 ROM image running at different speed in RAM may result in timing difficulties, but nobody I spoke to on the subject has any experience of this.

THE WIRING

QL network cabling is delightfully simple. A simple two wire lead (speaker or bell wire is perfectly adequate) with mono 3.5mm audio jacks at both ends is all you need to join up two QLs. Any reasonable length of cable seems to work, although I don't know if there is a maximum recommended distance between machines. As each QL (or Aurora or QXL) has two sockets, you simply daisy chain up to 63 machines together. The two machines on the ends of the network both have an unconnected socket, which I think has a resistor or something inside to terminate the network when no cable is inserted. In other words, you do not need to connect the unused end sockets of the two machines on opposite ends of the network into a loop or anything like that.

I bought a cable from my local TV and Hifi shop. This turned out to be a stereo 3.5mm lead (the QL sockets use the mono 2-pole version) which luckily seemed to work fine - I was a little bit afraid that the third conductors might cause a problem but it seemed not. Again - tested on QL only!

TOOLKIT 2

Tony Tebby's Toolkit 2 may be either in the form of a plug in EPROM (which uses the EPROM socket at the back of the QL), or built into disk interfaces such as those from Miracle Systems - Trump Card, Gold Card, Super Gold Card. Just about any QL system with a network socket apart from an unexpanded 128K QL will have a Toolkit 2 on board. In some cases it needs to be brought to life with a TK2_EXT statement in your BOOT program.

WITHOUT TOOLKIT 2

Even if you do not have Toolkit 2, a very basic level of operation is possible by using the NETI and NETO device names. These blindly send files to the network station numbers given, for example, if the first QL is station 1 and the second is station 2, the first can send a basic program to the second with the command SAVE NETO_2, while the second would receive it with the command LOAD NETI_1. The commands simply mean output this file to station 2, while the second simply inputs whatever was sent to station 2 from station 1. The NETO device name always requires the station number of the QL to which the data is to be sent, while the NETI device name requires the number of the station from which the data is being sent. Both should tie up if the file transfer is to succeed!

STATION NUMBERS

To identify computers on the network, each is given a number from 1 to 63. This is set with the NET command, and defaults to station number 1 if no NET command is issued. Issuing a NET 2 command sets that computer's network station number to 2.

On a simple 2-QL system (i.e. only two computers connected), both computers can have the same station numbers. It makes life simpler to have them both set to the same station number, indeed both can be station number 1 so on a simple 2-QL network, no NET commands need be issued at all!

Checking the network station number of a particular QL is not that easy. The network station number is stored in the system variables but there is no function to return the value so you have to carefully PEEK it if you really need to find it. It's a 1-byte value stored at address hex 28037 (decimal 163895) on systems where the system variables are at the old QL address. Better to regard it as an offset of hex 37 or decimal +55 from the base of the system variables. Here is a suggestion of how to allow for this on a Minerva or SMSQ/E system using VER$(-2) to find the base address:

```
9000 DEFine FuNction My_Station_Number
9010 LOCal v$
9020 v$ = VER$
9030 IF v$ = 'JSL' or v$ = 'HBA1' THEN
9040 REMark Minerva or SBASIC
9050 RETurn PEEK(VER$(-2)+55)
9060 ELSE
9070 REMark other systems
9080 RETurn PEEK(163895)
9090 END IF
9100 END Define My_Station_Number
```

Numbering of stations is not done automatically on a multi QL network - each machine has to have its number set with a NET command. It makes sense to number them from 1 upward starting from the first machine, although there are some considerations such as 'file server' machines having to have low station numbers (no higher than station 8).

BROADCASTING

Station 0 has a special significance - this is the 'broadcast' station. No physical QL on the network can have this number, but anything sent to station 0 can be picked up by all machines 'listening' to station 0. So if you want to send your basic program to everyone on the network, shout out 'all load from station 0 now' so that everyone can issue a LOAD NETI_0 command and you issue a SAVE NETO_0 command from your machine.

Another special station number is your own station number. If you 'listen' (or NETI) from your own station number, you can input from any station number. Thus if I am station 2 and I enter the command LOAD NETI_2 my QL will accept input from any station which happens to be sending me a file at the time. So you can either explicitly get a file from a particular station, or by listening to yourself you can accept from any station (e.g. if you know a file is to be sent to you but you don't know who is sending it from where!

There isn't really a lot more you can do on a system without Toolkit 2. You can transfer files in a very simple minded version, but you can only specify a network station number at the sending and receiving end, not a specific device on that station, so you have to use explicit commands on both machines, for example if I wanted to copy a file called FLP1_EXAMPLE from station 2 to ramdisk on station 3, I'd have to issue the following command on the sending machine (station 2)

```
COPY FLP1_EXAMPLE TO NETO_3
```

And on the receiving machine I'd have to enter a command like:

```
COPY NETI_2 TO RAM1_EXAMPLE
```

But once you have Toolkit 2 facilities things become much more versatile, although somewhat more complicated.

* You can have a file server machine (one whose drives and printers may be shared by other machines)

* You can not only send to and receive from specific station numbers, but also from specified devices and files on those stations. In other words, users do not need to spend so much time entering commands to specify what to do with the data once it arrives.
* You can do useful (if annoying) little things like open windows on a colleague's screen to send him/her a message, even get them to reply (who needs to email across a busy office!)

Toolkit 2 provides a program called a File Server which lets other QL stations access its drives, windows, printers etc. The file server is started with a simple command called FSERVE. This starts a little job running which takes care of handling the files passed to and from its machine. Once you have issued an FSERVE command you will find that if you examine the jobs list with a JOBS command, there will be a program called 'server' which may be removed with a command such as RJOB 'server' if you wish to stop it for any reason.

If you want to have more than one QL running a file server, this is perfectly possible, as long as you adhere to a simple rule - servers should only run on stations with numbers from 1 to 8. Any of the 63 possible stations can access these 'server' QLs, so taken to its logical extremes, you could probably have a 63 station office, with only 1 printer and hard disk between the lot I suppose. On a typical 2 QL network, you can issue the FSERVE command on both QLs and both can then access the other's drives, windows and printers.

Note: you should always set the network station number first, THEN issue an FSERVE command. I'm not sure why, it seems that FSERVE only looks at the network station number when it starts.

The device names for referring to devices on a file server is slightly different to the NETI and NETO names mentioned already. The new device consists of a name starting with the letter n and a station number, then an underscore and a full filename or device name.

SOME EXAMPLES

A 2-QL NETWORK

We have a 2 QL setup. We decide to leave them both with the default station number 1 (i.e. both are NET 1). FSERVE has been activated on both.

Either QL can now enter the command DIR n1_FLP1_ and it will give a list of files on FLP1_ on the other QL.

DIR n1_win1_programs_ will print a list of files from the directory called programs_ on the WIN1_ hard drive on the other QL.

COPY_N flp1_myfile_txt TO n1_par will copy a file called myfile_txt on FLP1_ on the first machine to the PAR printer port on the other QL.

WCOPY WIN1_programs_ TO n1_win1_programs_ will copy all files from a directory called 'programs_' on WIN1_ on one QL to a similarly named directory on WIN1_ on the other computer.

RENAME n1_win1_programs_myprogram TO n1_win1_programs_anothername will rename a program called 'myprogram' in the directory called 'programs' on WIN1_ on the other computer to a program called 'anothername' in the same directory on that computer.

OPEN #3,n1_scr_448x200a32x16:LIST #3:CLOSE #3 will list your current BASIC program to a window on the other computer.

MORE THAN 2 QLs

Suppose we have a three station QL system. All three have servers running. Station 2 wants a list of files on drive WIN1_ of station number 1. So station 2 enters the command:

```
DIR n1_win1_
```

Which gives him/her a list of the files held on win1_ on station number 1. Meanwhile, QL station number 3 has no printer, so wishes to print a listing of his BASIC program to that nice new printer connected to SER1 on station 2. So he enters the command SAVE n2_SER1. Or he could enter these commands:

```
OPEN #3,n2_ser1:LIST #3:CLOSE #3
```

Sadly, the printer (or probably the person) on station 2 does not deliver the printout to you, but that's life.

Meanwhile, the girl on station 1 wishes to remind her friend on station 3 that it's lunchtime, so she decides to send her a message to appear on her screen on station 3 and invite a reply from her:

```
OPEN #3,n3_con_512x256a0x0 : REMark full screen window on station 3
INPUT #3,'Hi, it's Linda on station 1, are you coming to lunch now? ';a$
PRINT a$
CLOSE #3
```

OK, that was a pretty silly example, especially as she wiped out her colleague's entire screen with the CON channel she opened, but it illustrates the kind of things possible. Messaging is usually better if you use somewhat smaller windows on the remote screen, just big enough for what you want to do rather than cover the entire screen. Also, be aware of the default colours when you open a window (green ink on black paper), so you may need some INK, PAPER, BORDER, STRIP and CLS commands to handle the window as well.

SOME MISCHIEF

On the basis that injecting some fun into networking helps to explain and stimulate interest, here are some of the things my grandchildren learned to do to me in order to annoy me while I was working away on my QL.

The example above illustrates one possibility!

```
OPEN #3,n2_con_512x256a0x0:PAPER #3:CLS #3:PAUSE:CLOSE #3
```

Screen goes blank. Grrrr. Has my QL crashed?

How to slow down a computer by keeping its server busy. Suppose that in revenge I (station 2) want to slow down the grandchild's QL on station 1:

```
REPeat loop
COPY RAM1_afile_txt TO n1_RAM1_afile_txt
PAUSE 5 : REMark optional
DELETE n1_RAM1_afile_txt
END REPeat loop
```

Thought the QL couldn't get a virus?

```
COPY FLP1_QUILL TO n2_WIN1_VIRUS
OPEN #3,n2_CON_128x64a0x0
CSIZE #3,2,1
PRINT #3,'QL ANTI-VIRUS'
PRINT #3,'Virus Alert On Your WIN1_ Drive!'
PRINT #3,'Check WIN1_VIRUS'
PAUSE : REMark goes away on pressing a key
CLOSE #3
```

That last one got them a right ticking off! Fortunately, as far as I know, you can't copy an executable program to another station and start it running (no doubt someone will prove me wrong!) so there's no chance of a QL virus spreading from computer to computer!

We made all these little programs up on the hoof as we went along and so I hope I've remembered them correctly.

Most commands which can take a channel number can do this sort of thing over the network to suitable devices, such as SAVE, SBYTES, OPEN, OPEN_IN, OPEN_NEW, INPUT, PRINT, LIST, DIR, INPUT, CLOSE and even functions such as FLEN can do this. A tribute to the QL 'device independence'.

The only things you strictly can't do is to set a fount over the network and anything which involves sd.extop (extended operations) - if you don't understand what sd.extop means (like me) you probably don't need to know. Just don't try to change founts over the network and you can probably do just about anything else you're likely to need to.

Another slightly more advanced little example: The file server is station 1. Station 3 wishes to access a modem connected to SER1 on station 1. Provided the user on station 3 (or the

software perhaps) knows how to control a modem with command strings, I'm sure it would be possible to access that modem from another machine, by opening a channel to the modem device and sending the relevant control bytes, which opens up some interesting possibilities once soql or any other QL internet software is a fully working reality.
Another possibility for you the readers to investigate as I haven't got the necessaries to test this theory of mine is extending the QL network with sernet to allow machines without QL network sockets to join the network. Sernet links computers via the serial port and uses device name 'S' in a similar fashion to 'N' for the QL network. I just wonder with both sernet and QL network file servers running if one network can access the other, e.g. station 4 is a PC running QPC2 (sernet station 2)
Station 1 is connected to station 4 with sernet link
Station 2 wishes to access something on QPC2 on station 4
What I'm wondering is: could station 4 enter something like
DIR n1_s2_win1_
If someone has the necessary hardware, please try this for me!
Even if you can't DIR n1_s2_win1_ I wonder if you could do something to mask the 'double networking' using either a DEV device to 'hide' one of them or an NFS_USE command (more details below)?
One extra little note, concerning Quill and other Psion programs for the QL.
Most programs which need to print to a printer on a file server machine can accept a printer device name such as n1_SER1 or N1_PAR. Quill and a few other programs can only do so indirectly, if you precede the device name with an underscore to imply that it's a non-directory device (i.e. the name you enter is not a filename). So in Quill, while printing to PAR or SER1 works OK of course, to print over the network you have to use a name like _n1_PAR. I have absolutely no idea why this should be so, I remember reading about it somewhere some time ago and it seems to be correct. The only thing I can think of is that if you try to print to N1_PAR it assumes this is a filename, so prints to a file called N1_PAR rather than to a printer on the PAR port of network station number 1, so the leading underscore in _n1_PAR is used to flag the fact that it's not to be interpreted as a filename.
What happens on a QL without the FSERVE file server running?
You can still access the network, but other stations can't access you. So if you have a printer connected to your computer, but no server running on it, nobody else can print to your printer. But you can still send stuff to the network, as long as it's to a QL with an FSERVE job running on it (and remember that FSERVE stations need to have station numbers as low as possible, certainly no higher than 8 according to section 22.2 of the Toolkit 2 manual.)
You can also still use the NETI and NETO device name even if the FSERVE job isn't running on your computer.
If there are three computers, but only station 1 has a file server running, it should be obvious that station 2 and 3 cannot access each other's drives, screens, consoles, serial and parallel ports with the Toolkit 2 n2_ and n3_ devices, although they may still be able to use NETI and NETO to send data between themselves to some degree.
Seasoned networkers will notice a slight problem with networking in general on the QL, in that you can either access everything on a machine, or nothing at all. There is no concept of 'shared folders' or anything like that available as far as I know, where you can effectively tell the rest of the network 'you can access my printer but not my floppy disk drive' or 'you can only access a few of the directories on my hard disk, the rest are private to me.'
That said, the QL network is still very useful either in an office environment or a home environment with more than one computer, but where it is preferable for only one machine to have a large hard disk and printer, for example.
Another potential problem with the QL network is that if there is a user on a file server machine, they may well experience a slow down while another user accesses their machine over the network. This may prove annoying if other users frequently access slow drives, e.g. save large files to your floppy disk drives where your QL may appear to freeze up until saving is completed. This became obvious to me when my grandchildren spotted that copying files to my computer slowed it down (just to annoy me). So you may find it better to dedicate one machine as a main server, without a user as such.

STORING PROGRAMS ON A SERVER

It is possible to store programs on a server and have EXEC commands load them from that device, by making use of the inbuilt default devices system provided by the DATA_USE, PROG_USE, SPL_USE and DEST_USE keywords.
It is quite acceptable to specify a network path such as n1_win1_programs_ in a PROG_USE statement, for example, so if you store your major programs in a directory called 'programs_' on win1_ on network station 1, all other stations could apply a PROG_USE n1_win1_programs_ statement, then any EXEC or EXEC_W commands should find the programs on the default drive, e.g.
EXEC MYPROG_exe
would do the equivalent of:
EXEC n1_win1_programs_MYPROG_exe
There might potentially be a problem if that program needed to load any configuration or data files, but most programs can be configured to know where their files live.
DATA_USE n1_win1_datafiles_ could do the same for data files and SuperBASIC programs. If that statement was issued on all other QLs, any commands which make use of the data_use default setting would route files to that directory on the server.
SAVE myprogram_bas would then save the basic program to the server. Where this may be most useful is if the server is the only machine having a hard disk, so everyone can make use of it. Not all programs use the DATA_USE defaults, though, and some may need to be specifically configured to make use of it, so DATA_USE may prove to be less useful in this context than PROG_USE.
Print spooling can be useful in this respect. If you have ever used the SPL or SPLF commands to send a file to the printer as a 'background job' you'll know how it can free up a QL by doing the copying to the printer in the background. On some systems, using a COPY_N command to send a file to the printer means that you are 'locked out' of SuperBASIC until it's finished, whereas SPL sends the file in a more multi-tasking fashion, allowing you to resume use of BASIC before the whole file has been sent. By setting the SPL_USE default to a name which does not end with an underscore (it's treated as a filename or directory name if a default ends with a '_' character) this is where the printed output is sent, so if the server is QL station number 2, and that has the latest all singing all dancing posh printer connected to the PAR printer port on it, everyone could use it by setting SPL_USE n2_PAR on their own machine. From there on, issuing a command such as SPL flp1_myfile_txt would send that file to the printer on station 2's PAR port. All stations would need to issue the command - it applies only to the machine on which it was entered. An SPL_USE command does not affect the entire network (nor PROG_USE nor DATA_USE for that matter).
Having tried the SPL related commands, you may like to experiment to see if the DEST_USE command can be used to provide default destinations for COPY and COPY_N commands in the same way. One word of warning, be careful with destinations, since you may confuse the wild card toolkit commands such as WCOPY if you set the default destination to a non-directory device such as PAR or SER1 or SCR or CON. Remember that names ending with a '_' character are directory names (i.e. handling filenames on that device) and any name which ends without a '_' character is a non-directory device such as PAR, SER1, SCR or CON which do not normally allow filenames.

NFS_USE

Not every program can access the network using the 'n' device name. Some QL programs can only have short and simple printer device names such as SER1, SER2 or PAR. Some software can't save files over the network using a compound name of the form networkstation_drive_filename_extension (e.g. n1_win1_filename_ext). Some programs are limited to simple filenames such as filename_txt - Quill, Archive, Abacus and Easel normally only allow DOS-style 8.3 filenames, meaning that the body of the filename should be no more than 8 characters long, then an underscore and finally an extension no longer than three characters. A filename such as ninechars_extn would be too long to handle! This does not apply to the Xchange version of these programs in the same way.
So the designer of Toolkit 2's network extensions thoughtfully provided the NFS_USE command, to 'hide the network from applications by setting a special name for a network file server' (I quote from the Toolkit 2 manual).
If you are familiar with the DEV or SUB devices, NFS_USE should be fairly simple to understand.

The command has this syntax:

NFS_USE device,drive1_alias,drive2_alias, etc up to drive8_alias

'device' is a 3 character name such as MDV or FLP which is to be masked to appear as an equivalent device on the network server.

Let us assume that station 1 is our file server. Station 2 has some awkward old software which only knows about microdrives, not floppy disks or hard disks, let alone network and file servers.

Station 2 does not use microdrives (does anyone use microdrives these days?), so what we could do is to re-map the device name MDV to refer to the file server. We need to decide where MDV1_ is to refer to, where mdv2_ is to refer to and so on. In this case, let us assume that MDV1_ will be in a directory called FRED1 on win1_ on the server (n1), and MDV2_ will be in a directory called FRED2_ on win1_ on the same network station number.

We could decide something similar for all 8 possible microdrive numbers, but programs rarely access higher than MDV2_, so we'll stick with two devices for simplicity.

NFS_USE mdv,n1_win1_fred1_,n1_win1_fred2_

The above command is entered on the QL station 2, not on the file server machine.

What it means is that every time a program tries to save to MDV1_, it is fooled into thinking that n1_win1_fred1_ is really MDV1_ and that n1_win1_fred2_ is really MDV2_. As in previous examples, if the paths end with an underscore '_' character, they are assumed to be directory devices.

You need to be careful with other xxx_USE commands such as FLP_USE when you use NFS_USE, to avoid possible confusion arriving. Some disk interfaces may try to access their own floppy disks before realising that you've used NFS_USE to try to map them onto the server's hard drive. So once you've set the NFS_USE command to 'hide' the floppy disk drives, you may also need a FLP_USE 'FDK' command to rename the FLP1_ and FLP2_ disk drives to something like FDK1_ and FDK2_ to work around this!

As I write this, I realise I have not tried non-directory devices with the NFS_USE command, so if MDV3_ in the example above was set to n1_SER1 I don't know if it would be allowed or how well it would work if you tried to PRINT to MDV3_! One for you to try out.

Another exercise for you to try would be to use NFS_USE with non-directory device names purely for printing purposes. Assume station 2 has a broken printer, so nothing connected to SER1 or SER2 or PAR. Station 2 realises that stations 1, 3 and 4 all have printers and so tries to use an NFS_USE command to remap his SER1 and SER2 to other users. Would this work? Try it for yourselves!

NFS_USE 'ser',n1_PAR,n2_PAR,n3_PAR,n4_PAR

Could he now print to SER1, SER3 and SER4 and so have a choice of printers?

NETWORK APPLICATIONS

After looking around, I found that there are some QL network utility programs out there in PD libraries, programs like Netpal and Flexynet. A subject for a follow up article at some point in the future I suppose.

OK, that's it. I'm off down the pub to do some networking of a different kind.

TOOLKIT II TUTORIAL

Stephen Bedford

TOOLKIT II is most likely, the widest held product ever developed for the QL. It can be found on disk interfaces, on ROM, and a configurable version on disk. It surely is a "must have" product, as it makes life a whole lot easier. As with most good things, there is something lacking, in this case its the documentation. The purpose of this TUTORIAL is to make additions to the current documentation.

As the section numbers are those used in the TOOLKIT II manual, numbering may appear inconsistent when a section has deliberately been left out, this is because it is sufficiently covered in the manual

INTRODUCTION: Tool kit II (hereafter referred to as TKII) is a collection of over 120 additions to SuperBasic, these fall into two categories: those that extend the capabilities of SuperBasic as a programming language, and those that enhance SuperBasic's role as the command language of the QL. The manual supplied with TKII details all of the commands and functions available, therefore rather than explain every feature in detail, these notes are written as additions to the existing documentation.

Tony Tebby divides the facilities of TKII into three categories: the two alr%adv described above, and the third being that of development facilities. This latter category consists of just two commands, which nonetheless transform the QL. The manual explanation of these commands in section 3 is fairly complete, however brief notes follow.

3.1 ED - SuperBasic Editor

ED is a SuperBasic window-based editor (you can use any portion of the screen required by setting the size of #2 appropriately or by using another channel, like #3, and using the command ED #3). Note that such a channel should be opened as a console device, not a screen device, rather use OPEN #3, con. This is the case whenever a channel is required to accept input as well as output to the screen. A console device is the combination of a screen window and a keyboard queue.

3.3 Viewing a file

View is used to display a file. It is similar to TYPE as used in many other operating systems (MS-DOS).

COMMAND LANGUAGE EXTENSIONS: The QL is an exceptional

computer because of its inbuilt software: both the operating
system QDOS, arid its programming language SUPERBASIC. QDOS
even now, has no rival among operating systems found on
affordable micro-computers while many of the advanced
features of SUPERBASIC have yet to be seen in any other
implementation of the language. On a machine like the QL,
BASIC isn't just a programming language, but is actually the
command language of the computer. Without resorting to
machine code, it's possible to run several jobs concurently,
but not possible to alter the priority of these jobs, or to
remove them or to see what jobs are currently on the machine
or to see what state they are in (active, suspended, or
inactive). It's also not possible to see how much free
memory is available on the machine without using PEEK. TKII
rectifies these and manymore deficiencies of SuperBasic.

4. DIRECTORY CONTROL

Directory Control is an area that should be looked at in
greater detail. There are two main difficulties when trying
to understand this. The first is attitude, being that if
something is different from IBM it must be wrong. The second
is the complexities of wildcards, as used in QDOS.
The first can be overcome with time, as you use the features
of QDOS (TKII should be considered an integral part), and
coming to the realization that it's a superior operating
system compared with MS-DOS. The second is best overcome by
use of examples to illustrate the use of wildcards, and by
practice in their use.

4.1 DIRECTORY STRUCTURES

Refer-to the TKII manual, and note examples below.

4.2 SETTING DEFAULTS

QDOS provides the user with the facility to set three
directory defaults, using the commands: DATA_USE, PROG_USE,
and DEST_USE. For an unexpanded machine the defaults when
the machine is booted are: MDV2_, MDV1_,and SER
respectively. On a machine with floppy disk drives the
defaults are: FLP1_, FLP2_ and SER.
The DATA_USE default is used for most filing system commands
such as: LOAD, LRUN, MERGE, LBYTES, SAVE etc. Thus you could
set the default for data as follows: DATA_USE FLP1_BASIC.
Suppose the disk in FLP1_ had the following files on it:

```
basic_mandelbrot_bas
basic_prime_bas
basic_game_pas
pascal_trig_pas
pascal_trig_rel
pascal_trig_bin
pascal_game_bas
pascal_game_rel
pascal_game_bin
letter_qjump_txt
address_qjump_txt
```

After setting the data default as above, a directory listing
using DIR would look like this:

```
basic_mandelbrot_bas
basic_trig_as
basic_game_bas
```

While DIR FLP1_ would show the full contents of the disk, if
you wished to know about all PASCAL associated files, you
would set the default to FLP1_PASCAL_. If you wished to set
only the games related files, you could set the default as
follows: DATA_USE FLP1_GAME_. A DIR then would show the
following:

```
basic_game_bas
pascal_game_bas
pascal_game_rel
pascal_game_bin
```

The data default has been set with WILD CARDS. The string
making up the default may be broken up into four parts:
- "FLP1_" is the device name.

- "_" is a wild card that represents any characters up to the point where the next part of the default matches.
- "GAME" is the portion that follows an underscore (since this is where the proceeding wildcard ends). This might be considered to be the file name.
- "_" this is a second wild card, and could be considered to mean, that any extension (conventionally indicating file type) is valid.

Thus the default is set to FLP1_ followed by any characters up until a match with the next part of the default (in this example 'game') by 'game' followed by any characters. That is any files of any type, called game on any directory on the disk. The idea of directory, subdirectories, file name and extension, are used as an analogy with other operating systems which many people are familiar. Perhaps it's better to think of these as discrete parts of a file name.

Remember, the first part of a full filename is always the device name, and it's a good practice to have the last part (extension) indicate the type (bas for BASIC files pas for PASCAL etc.).

An underscore therefore represents both a deliminator between the parts of a default and wildcards. The general rule being that a single underscore within a default only acts as a deliminator, two underscores within a default represent one deliminator and one wild card. A single underscore at the end of a default may be considered as both a deliminator and a wild card Thus, in the example, a default of FLP1_GAMES_ would give the same directory listing, Whereas FLP1_GAMES would show no files at all. We haven't supplied a wild card, and there are no file names that start FLP1_GAMES. Note that an underscore is automatically appended to a default if it doesn't already end in one.

If we were to set the default back to FLP1_BASIC_ , we could load the mandelbrot program simply by typing:

LOAD MANDELBROT_BAS

It appears that for commands other than DIR, wiidcards within defaults don't work. That's to say the name appended to a command such as LOAD is tagged onto the end of a default rather than replacing a wild card. This avoids possible ambiguities. Also commands other than DIR need some part of the file name appended to them. One cannot set the default to FLP1_BASIC_GAMES_BAS and type LOAD. This will result in a 'bad parameter error'.

PROG_USE is used to set the directory for executable code and as such is only used for: EXEC, EXEC_W and the TKII commands EX ,EW, and ET. Thus continuing the example used above, you could set the program default as follows:

PROG_USE FLP1_PASCAL

and then execute one of the programs as follows: EX TRIG_BIN. Notice that the TKII command EX may be used exactly as the standard EXEC but as shall be seen later, can also be used in different ways.

DEST_USE sets the default destination for commands such as COPY and RENAME (TKII). This is by default, set to SER1 so that using COPY with only one parameter will result in a file being printed, if a printer is connected via SER1.

4.3 DIRECTORY NAVIGATION

The commands DDOWN, DUP, and DNEXT provide another method of changing the data default (and the program default if it's the same as the data default). These commands allow the default to be altered relative to the current value as opposed to setting the default in an absolute manner as with DATA USE (look at commands LINE and LINE_R page 32 of Keywords in the QL User Guide for an analogy). Under an operating system such as MS-DOS with its single default directory, a single command is used far changing the directory both relatively and absolutely.

The command DDOWN allows one to move further down the

directory tree. That is, extend the data default. An
underscore is automatically attached to the argument
appended to DDOWN. Assume the machine has just been booted
so that the data default is FLP1_, following the same
example used above, if you wished to look at only PASCAL
files you could set the data default using the command:
DDOWN PASCAL. This is then equivalent to the statement:
DATA_USE FLP1_PASCAL. Now if you wanted to see only PASCAL
source files, you could type the command: DDOWN_PAS. The
underscore is used as a wild card. You wish to see all files
with the _PAS extension in the directory PASCAL.
The data default is now: FLP1_PASCAL_PAS_.
The command DUP doesn't take any arguments, it moves the
data default up one directory level. In this case DUP would
set the default back to FLP1_PASCAL.
DNEXT allows you to move to a different default at the same
level, if the current default is FLP1_PASCAL,PAS and you
used the command: DNEXT REL, the default would become:
FLP1_PASCAL_REL_. You could then list the REL files (output
from compiler). If you wished to set the default for the BIN
files (output from the linker) you couldn't type DNEXT bin.
This is because BIN is an extension provided by TKII for
using binary numbers, therefore you would have to type:
DNEXT 'BIN'. This prevents SuperBasic from evaluating the
argument, and is true of SuperBasic commands in general, not
just those explained here.

4.4 TAKING BEARINGS

One command and three functions are provided to allow you to
find out the current
defaults: DLIST, DATAD$, PROGD$, and DESTD$.
DLIST lists the current defaults in the order data, prog and
dest. As usual a channel number may be appended if one
doesn't wish the output to go to window #1. As with other
commands added or modified by TKII, an implicit channel may
also be used: DLIST #2, DLIST \FLP1_defaults.
The three functions each return the value of the appropriate
directory. For example: PRINT DATA$, might result in
FLP1_PASCAL_BIN being printed to the screen.

5. FILE MAINTENANCE

TKII improves file maintenance procedures in two ways.
First, the existing commands COPY, DELETE, and DIR now use
the default directories and secondly,the addition of wild
card and overwrite operations significantly ease file
handling.

5.1 WILD CARD NAMES

The manual says that wild card characters are not used,
rather any missing section of a name is treated as a wild
card. However, in part one of this tutorial i stated that an
underscore is a wild card character. In fact,these two views
are essentially the same, but considering the underscore to
be a wild card is easier to understand. The use of wild
cards in this section is the same as used for default
directories explained in the first part (section 4.2).
However for the notes to be correct a stricter definition of
what the underscore can represent must be defmed:
AN UNDERSCORE CAN REPRESENT A NULL STRING OR ANY SERIES OF
CHARACTERS THAT DO NOT START WITH A DELIMINATING UNDERSCORE
AND END WITH EITHER AN UNDERSCORE OR THE END OF THE NAME.
This is consistent in many cases with saying that the
missing section is treated as a wild card. The following
example explains the definition. Suppose you have a disk in
FLP1_ with the following files:
BASIC_MANDELBROT_BAS
BASIC_GAMES_BAS
BASIC_JOBS_BAS
BASIC_PROGRAM_BAS
If the data default is set to FLP1_ (DATA_USE FLP1_) then
DIR BASIC_BAS would show: BASIC_PROGRAM_BAS. The underscore
is representing PROGRAM_. The other files do not match since

the underscore following BASIC is a deliminator, and the
wild card cannot represent a string that starts with an
underscore. However, DIR BASIC_BAS would show all the files.
The first three file names the first underscore in the wild
card name is the deliminator, while the second represents:
MANDELBROT, GAMES, and JOBS_ respectively. For the last file
name, the first underscore is set to a null string, and the
second matches PROGRAM_ as before.
Thus, a wild card name of FLP1_BASIC_BAS could match a file
name of: FLP1_BASIC_MANDELBROT_BAS , and it may be
considered that either MANDELBROT is the missing section of
the filename, or that the second underscore in the
wild card name matches MANDELBROT in the file name.
If a disk contains a file with the name letter on it, then
the command DIR 1_ will result in the file letter being
listed. Yet an underscore does not appear in the filename,
suggesting that the underscore is a wild card matching a
series of characters ending with the end of the file name.
In section 4.2 of the manual it explains that if a default
directory is set that doesn't end with an underscore, then
an underscore is automatically appended. This may be
considered the case for wild card name too. Thus in this
example DIR 1, would also result in the file name letter
being displayed.
It doesn't really matter how wild cards are defined, the
important thing is to realize that they are very useful.
Practice in the use of wild card names will hopefully bring
understanding.

5.2 DIRECTORY LISTING

As well as the standard DIR command TKII also makes
available WDIR, and WSTAT. All use the default data
directory and may be passed wild card names. The output of
the commands may be redirected using implicit channels as
shown in part 1. If you have a disk in drive one with a name
TKII NOTES and the following files:

```
TKIIa_DOC
TKIIb_DOC
TKIIjob_DOC
```

Then a DIR FLP1_ would give you the following display:

```
TKII NOTES
1347/1440 sectors
TKIIa_DOC
TKIIb_DOC
TKIIjob_DOC
```

WDIR FLPI_ would give you:

```
TKIIa_DOC
TKIIb_DOC
TKIIjob_DOC
```

WSTAT FLPI_ would give you:

```
TKIIa_DOC
16590 1990 Jul 06 20:25:30
TKIIb_DOC
17065 1990 Jul 10 15:53:29
TKIIjob_DOC
8412 1990 Jun 23 17:16:57
```

Notice the amount of space on the disk is shown in sectors
(blocks of 512 bytes). The file sizes are shown in bytes,
however the space for a file is allocated in groups of
three sectors, thus TKIIa_DOC would use 33 sectors,
TKIIb_DOC would use 36, and TKIIjob_DOC would use 18
secto;rs. That is 87 sectors in all, the other 6 sectors
that have been used are for the directory and map (a
directory of a blank disk will show 1434/1440 sectors).
WSTAT is very slow on microdrive.

5.3 DRIVE STATISTICS

The command STAT, shows Just the name and space available on
a disk. In the above example the display would be:

```
TKII NOTES
1347/1440 sectors
```

To get full information on a disks contents type:
STAT FLP1_ : WSTAT FLP1_
Note in the contents section of the TKII manual a command ASTAT is mentioned. This command, which should produce an alphabetic list of files, is not described elsewhere in the manual, and is in fact, not implemented in the versions of TKII that I own (versions 2.12 and 2.13)

5.4 FILE DELETION

THE DELETE command has been modified to use the data default directory. Thus, for a machine with floppy disks attached, just after booting, the command: DELETE BOOT would delete your boot file contained on FLP1_. For a microdrive only system the same command would try to delete a file named boot on MDV2_.
A new command has been introduced, WDEL, this command will accept wild card names as a parameter. Suppose you are using the same disk as above containing the files:
TKIIa_DOC
TKIIb_DOC
TKIIjob_DOC
Typing the command DELETE TKII would result in the disk spinning and no error message would be produced, yet nothing would be done: the file TKII doesn't exist. Whereas the command, WDEL TKII would produce the following response: "FLP1_TKIIa_DOC..Y/N/A/Q" meaning, is this file to be deleted (YES or NO), are ALL files that fit this wild card to be deleted, or is the operation to be QUIT. So, to delete all except the first of the files that fits the wild card, first respond with N then A. It's suggested that the option of deleting all matching files is not used until familiar with this command and wild cards. On a machine running MS-DOS a prompt similar to the one described above, is not given, DEL TKII would go ahead and delete all files that match.

5.5 FILE COPYING

The standard COPY command has been modified to use the DATA and DEST default directories. Thus, the command; COPY BOOT, will copy FLP1BOOT to SER1. That is, assuming the defaults have not been altered. So, if a printer is attached, the file will be printed
A further alteration to the COPY command is that if the destination file already exists, permission to overwrite is asked for. Thus typing: COPY TKIIa_DOC to TKIIb_DOC, the following prompt would result: "FLP1_TKIIb_DOC exists, OK to overwrite..Y or N" This is very much like the QUILL SAVE operation.
The COPY command has become more "intelligent". The file header is automatically either copied (making a copy of an executable file) or not (printing a file) depending on the file and devices concerned. I haven't used COPY_N or COPY_H since having TKII.

5.5.1 SINGLE FILE COPIES

This includes the standard COPY command as described above, COPY_N and COPY_H which have also been modified to use default directories, and COPY_O. In my copy of the TKII manual, a misprint has lead to COPY_O appearing as COPY_.
The COPY_O command will copy a file without asking what to do if the destination file already exists. This is useful when copying is performed within a SuperBasic program and one does not wish to give the user the choice of whether to overwrite a file or not.

5.5.2. WILD CARD COPIES

The command WCOPY allows you to copy a number of files as a single operation. As with the commands for single file copying, WCOPY uses the default directories. The form of the command is:
WCOPY #channel, source TO destination
As with standard QDOS commands the channel is optional but if supplied it is where the prompts will be sent. If a

channel is not specified, then prompts will be sent to
#0.
The following examples illustrate the use of the command.
Assume the data default directory is set to FLP1 _ and the
destination default directory is set to FLP2_ and that the
disk in drive one contains the following files:
TKIIa_DOC
TKIIb_DOC
TKIIjobs_DOC
LETTER_RICHARDALEXANDER_TXT
ADDRESS_ALEXANDER_TXT
for all of the examples.
i) WCOPY
This is the equivalent to WCOPY #O, FLP1_ TO FLP2_ That is,
copy all files from FLP1_ to FLP2_. However, as with the
WDEL command a prompt is given:
FLP1_TKIIa_DOC TO FLP2_TKIIa_DOC.. Y/N/A/Q
Responding with A would lead to all files being copied from
the first to second disk drive individual flies may be
selected for coping by responding - YES or NO as each
filename is presented. The operation maybe QUIT at any time.
ii) WCOPY #1, TKII_ TO NOTES_
This is the equivalent to WCOPY #1, FLP1_TKII_ TO
FLP2_NOTES_ Thus a selective copy of only files that are
notes on TKII is performed, arid would therefore produce the
prompt:
FLPI_TKIIa_DOC TO FLP2_NOTESa_DOC..Y/N/A/Q?
The part of the name represented by the wild card is
appended to the destination wild card name. For the files
that match this specification on the disk in FLP1_ the wild
card will have in turn the values a_DOC b_DOC, and Jobs_DOC.
The prompt will appear in #1 unless the windows have been
changed, at the top of the screen.
iii) WCOPY TO FLP1_BACKUP_
This is the equivalent to WCOPY #O, FLP1_ TO FLP1_BACKUP_
and allows copies of all files to be made on the same disk
but with a prefix added to the file name ie copy files to a
subdirectory.
iv) WCOPY TO SER1
This is equivalent to WCOPY #O, FLP1_ TO SER1, and will
result in an error: BAD NAME, because SER1_TKIIa_DOC is not
a valid name. If at any time the resulting destination file
exists already, a prompt asking if the file should be
overwritten is produced.
5.5.3 BACKGROUND COPYING
The command SPL is provided to allow background copying in
the same manner as COPY_O. The copying is performed by a
spooler which is an independent Job. The primary use for the
spooler is to print files. SPL uses the data and destination
defaults and so if the QL has Just been booted then you can
print a file as follows:
SPL TKIIa_DOC
The command doesn't accept wild card names. So that a file,
FLP1_PRINT_CMD (the extension_cmd shows that the file
contains a series of commands rather than a numbered
SuperBasic program - the use of sensible and consistent
extensions can greatly assist with file management. The
extension _bat may be chosen as with MS-DOS) could be
created containing the following lines;
SPL TKIIa_DOC
SPL TKIIb_DOC
SPL TKIIjobs_DOC
The command LRUN PRINTCMD would then allow the three files
to be printed withoit intervention while the machine can be
used for other things. Three separate Jobs would be created
all named SPL and all running at priority O. At the default
priority the background printing will have little effect on
ones main Job whether it be editing or playing a game.
However, if the destination is a file rather than the serial

port, this will not be the case. When spooling to a file, keyboard response will fluctuate considerably no matter at what priority the spooler is running. Spooling to a file will obviously be much quicker than spooling to a printer, but offers no real benefits over copying to a file.
The output for the spooler is selected using the command SPLUSE. This is in fact, the same as DEST_USE except that an underscore is not appended to the name - an underscore at the end would indicate a wild card name and SPL doesn't accept wild cards.
SPL_USE FLP1_DUMP would set the destination default to FLP1_DUMP and all subsequent uses of SPL would write to that file automatically overwriting the previous version. A variant of SPL, SPLF will spool a file and place a form feed at the end. This will ensure that individual files are printed on separate sheets of paper. Both of these commands may be supplied with channel numbers rather than filenames as explained in the TKII manual.
At this point it is worth mentioning one of the many wonderful features of TKII that I don't think appears explicitly in the manual. Although not directly connected with spooler it is to do with printing. If, on a QL without TKn fitted, there are two (or more) Jobs running, both of which are trying to access the printer, the result will be a printout which is a mess the output of the two Jobs interleaved. With TKII fitted, your Job's output will be sent to the printer while the other's is buffered in memory. Once the printer is free, the buffered output is copied from memory to the printer.

5.5.4 RENAMING FILES

As explained in the TKII manual the remaining commands follow the same form as the equivalent copying commands, but merely alter the filename ie RENAME has similar syntax to COPY and WREN has similar syntax to WCOPY.

6. SUPERBASIC PROGRAMS

6.1 DO

DO is a command for an executed SuperBasic command file, which is a file containing unnumbered BASIC statements. Thus, using the example from 5.5.3, the command: DO PRINT_CMD, would perform the three spooler commands contained within the file. The advantage of the DO command being that the current SuperBasic program is unaffected. It would be lost if you used LRUN. Any block commands within a command file must appear on a single line, for example:

```
FOR n = 1 TO 10: PRINT n
REPeat read: INPUT a$: PRINT a$, CODE (a$)
```

would be an acceptable file, whereas the following would not:

```
FOR n = 1 TO 10
PRINT n
END FOR n
REPEAT read
INPUT a$
PRINT a$, CODE (a$)
END REPeat read
```

An attempt to LRUN such a file would lead to the error "not found". This refers to loop control 'a' which only exists in the line of the definition. It is of course, acceptable to use either upper or lower case for keywords, and use normal abbreviations. Note the warnings at the end of 6.1 in the TKII manual.

6.2 DEFAULT DIRECTORIES

The normal BASIC filing commands have been modified to use the default directories. In addition the LOAD command will look for a file in the PROGRAM default, if it doesn't locate it in the DATA default directory. An overwrite variant of the SAVE command, SAVE_O has been introduced that works in the same manner as other overwrite commands.

7. LOAD AND SAVE

This section refers to the loading and saving of binary
files, i.e. LBYTES and SBYTES for resident procedures and
EXEC, EXECW, and SEXEC for transient programs. SBYTES and
SEXEC have been modified in the same way as other commands
that write to files prompt appears if file already exists),
and the overwrite variants have been introduced.
A new command, LRESPR, has been added that combines the
functions of RESPR, LBYTES, and CALL. Thus: base = RESPR
(file_length): LBYTES file, base: CALL base, can simply be
performed by typing: LRESPR file. With the latter it's not
necessary to explicitly find out the length of the file. As
with RESPR, LRESPR may only be used if no other jobs, other
than BASIC are running on the QL.

8. PROGRAM EXECUTION

This section deals with the commands for executing compiled
programs which run on the QL as jobs. This formerly
consisted of two commands EXEC and EXEC,W. These have been
modified and made synonymous with new versions: EX and EW.
Another command, ET has been introduced which loads a
program into memory but returns control to BASIC before
starting the job. The EX command is explained further to
illustrate the new facilities provided by all of these
commands.

8.1 SINGLE PROGRAM EXECUTION

EX may be used in the same way as the standard EXEC command
in order to start a job on the computer: EX filename. The
command will look for the file on the program default
directory. In addition, the program may be passed a
parameter string. As an example of use I'll refer to a
commercial program 'MASTER SPY EDITOR'. This program
can be invoked as follows: EX MS, FLP1_BOOT. This command
executes MASTER SPY (which I've renamed to MS on my working
copy) which loads the file FLP1_boot and presents it ready
for editing. This feature was made available on MASTER SPY
(version 1.7 and onwards) as a result of my writing to ARK
to ask if it were available.
A further feature of the EX command is that filename (or
channels) may be passed to a program for use as it's
standard input and output. BASIC programs compiled using
SUPERCHARGE cannot be passed input and output files, perhaps
TURBOCHARGED programs can, I don't know. But it is easy to
write a PASCAL program to accept filenames for input and
output channels, and is a standard feature of PASCAL. Below
is an example PASCAL program which should be quite easy to
follow for anyone familiar with SuperBasic. Comnients are
enclosed between curly brackets.

```
PROGRAM mu12 (input, output);
VAR
param : string [20]; { like DIM param$(20) }
in_num, out_num : real;
BEGIN
REPEAT
getcomm (param); { read the parameter string }
writeln @aram)
readln (in_num) { equivalent to INPUT a }
out num:= in num * 2
writeln (2 * in_num); { equivalent to PRINT 2*a }
UNTIL in_num = 0;
END
```

This program simply reads in numbers and writes out double
the number. If the program was invoked using EX MUL2_BIN
(the file mul2pas is passed to the compiler which produces
mul2rel [the extension .obj would be used on MS-DOS systems]
and then the linker processes this file and produces mul2bin
[.exe under MS-DOS]) then the numbers could be typed in at
the keyboard, and the answers would be printed to the
screen. Because no parameter has been passed only the
numbers would be displayed on screen, however the same
program could be invoked as follows:

```
EX MUL2_BIN, IN_DAT< OUT_DAT;'IN_DAT * 2'
```
MUL2 would have to be located in the program default and
IN_FILE in the data default directory. If IN_FILE contained
the following lines:
```
2.7
-3.34
10.6
O
```
Then the file OUT_DATA would be produced in the data default
directory containing the following lines:
```
IN_DAT * 2
5 .4000000E+00
-6.6800000E+00
2.1200000E+01
0.0000000E+00
```
The file OUT_DAT would be overwritten automatically if it
already exists. Note the numbers may be easily formatted so
as not to use scientific notation, this is merely the
default. The same results could be achieved by passing
channel numbers instead of file names:
```
OPEN_IN #3, IN_DAT
OPEN_NEW #4, OUT_DAT
EX MUL2, #3, #4;'IN_DAT * 2'
CLOSE #3: CLOSE #4
```
The Propero PASCAL compiler and the GST LINKER also accept
parameter strings. The compiler uses the parameter string to
pass the name of the PASCAL program and flags indicating
various options for the compilation. Likewise with the
linker one passes the program name and the name of the file
containing the linker directives. The parameter need not be
a string constant, it could be a variable:
```
FILES = FLP1_BOOT : EX MS;FILE$
```
8.2 FILTERS
EX also allows a series of programs to be executed that work
together to process a stream of data, the output from one
program being passed to the input of the next. The situation
is analogous to a production line. In the TKII manual it
explains that a series of programs (or filters) could be
executed as follows:
```
EX UC, FRED, TO LNO TO PAGE, SER;'FILE FRED' & DATES
```
Such a series of programs could be easily written in PASCAL
but the string handling is sufficiently different from
SuperBasic so as to make the example of little use. Instead,
consider a simpler set of programs:
```
EX ADD3_BIN, IN_DAT TO MUL2_BIN, OUT_DAT;'NUMBERS'
```
MUL2 is the same program as listed above. The output from
ADD3 goes to the input of MUL2 and the output goes to the
file OUT DAT. The file OUT DAT will have the heading
'NUMBERS'. The program ADD3 is as simple as MUL2:
```
PROGRAM ADD3 (input, output);
VAR
in_num, out_num : real;
BEGIN
REPEAT
readln (in_num);
out_num := in_num + 3;
writen (out_num);
UNTIL in_num = -3;
END
```
This program reads a series of numbers and writes the values
plus three. It stops when it reads the number -3, this will
have three added and be passed to MUL2 which stops when it
reads the number O. So they stop properly together. If any
program in the chain failed, then the whole series of jobs
involved would be removed.
Suppose that IN_DAT now contains the following lines:
```
2.3
-3.6
10
```

The ADD3 (the job name is derived from the name on the PROGRAM statement in the PASCAL program) will read this file and pass the following numbers to MUL2:

```
5.3000000E+01
-6.0000000E+01
1.3000000E+01
0.0000000E+00
```

MUL2 will read the numbers, and produce the file OUT_DAT:

```
NUMBERS
1.0600000E+01
-1.2000000E+00
2.6000000E+01
0.0000000E+00
```

The means of communications between these two programs is via a pipe. If the IN_DAT is a much bigger file say, a thousand lines, then while these programs are executing inspection of the channels menu in QRAM shows that there is a pipe associated with both of the programs.

Each of the programs in the chain may have many other channels open and use the screen and keyboard as well as other files and devices. However, if using software like QRAM, it is important to remember that if the programs in the chain are competing for the screen, then one will be suspended, this will cause the chain of programs to fail (the first program in the chain may be suspended, and this will suspend the chain of jobs once the pipes have been emptied). With the PASCAL programs as described, the programs will fail even though output is not sent to the screen. This situation may be remedied by using the UNLOCK utility supplied with QRAM.

I would think that it would be possible to write similar programs in FORTRAN, in which case unit 6 of one program would be attached via a pipe to unit 5 of the next. 'C' also has standard input and output, which I'm sure would accept pipes (on a full implementation of the language).

9. JOB CONTROL

The QL was the first affordable computer to allow multitasking. It's one of the many features that still sets it apart from the herd, and yet, is one of the most difficult to control satisfactorily on a standard machine. It is also one of the areas that most interests 'Tinkerers' like myself.

The extensions for job control are documented in section 9 of the Toolkit II manual. There are four commands (JOBS, RJOB, SPJOB, and AJOB) and four functions (PJOB, OJOB, JOB$, and NXJOB) provided.

9.1 JOB CONTROL COMMANDS

JOBS lists the currentjobs. By default the output will go to #0, but as with standard SuperBasic procedures, the output may be sent to any other channel by simply appending # and the channel number. Thus JOBS #2 will display a list of jobs in #2.

Toolkit II also allows implicit channels. That is, if you wish to send the output to a device, you need not open a channel to that device, send the the output to the channel (as stated above) then close the channel, but you may append the command with \ and the device name. The following will create a file JOB_TXT on flpl_ that contains the list of jobs on the system:

```
JOBS \flp1_job_txt
```

You can just as easily print out the list of jobs with one command:

```
JOBS \ser1
```

With SuperBasic as the only job in the machine, the JOB command would display a table as follows:

```
Job tag owner priority
0 0 0 32
```

SuperBasic does not have a job name. Names are normally

displayed after the job priority. If a job is suspended (see SPJOB below) then an 's' would be shown immediately to the left of the job priority.
RJOB allows you to remove a job (other than SuperBasic) from the machine. RJOB is followed either by the job name or by the job id. The job id is a combination of two parameters, the job number and the job tag. These values are displayed by using the JOB command. If you execute a program, mandelbrot, using the command ex flpl_mandelbrot, the JOB command would produce a list as follows:

```
Job tag owner priority
0 0 0 32
0 0 0 8 mandelbrot
```

Note that a job activated by SuperBasic will start with a priority of 8. Some jobs will not have names. The job mandelbrot may be removed as follows by the command RJOB mandelbrot.
There is a further parameter that may be added to the command. This is an error code which through the use of machine code could be read by the parent job (that job that started the job being removed). Thus you could type RJOB mandelbrot,-l. This isn't relevant to the user of SuperBasic except that it appears when you follow RJOB with the job id instead of the job name, the error code must also be used. Thus you would type RJOB 1,0,-1. The reason for using the job id as a parameter is that some jobs do not have names. Another reason is that it is possible to have many jobs with the same name. If the job name is used, then QDOS will remove the first of that name.
SPJOB allows you to set a jobs priority. As stated earlier, a job started by SuperBasic is given a priority of 8. If the job is required to run faster or slower then you must raise or lower its priority. Like RJOB the first parameter is either the job name or job id. If a job id of -1 is used then the current jobs priority is altered. (If you type SPJOB as a direct command it would be SuperBasic, however, if SPJOB is used in a SuperBasic program that is then compiled an argument of -1 would refer to the compiled program). The second parameter is the priority. This is an integer between O and 127. A priority of O means the job will become inactive (it will not get a share of cpu time).
Thus, if you wished to alter the priority of the job mandelbrot (as used in the example above) you could type either of the following:

```
SPJOB mandelbrot,l6
SPJOB 1, 0, 16
```

We would then expect the job to run approximately twice as fast. Note the speed of execution of a job not only depends upon its priority, but also upon the availability of resources it wishes to use. For example, only one job may use the keyboard at a given time, so if a job requires input and the keyboard is already in use, then it will be suspended by QDOS until the keyboard is available (either because the first job has finished or control-C is used).
AJOB is used to activate a program which has been loaded into memory but not previously started. (If a job has previously had its priority set to 0 it could be reactivated either by setting its priority to a positive value or by using AJOB. A job executed with the command ET would be in an inactive state until activated by further commands such as AJOB).

9.2 JOB STATUS FUNCTIONS

PJOB returns the priority of a job, as with the commands above, the job may be specified either by its name or by its id. Since it is a function the job name or id must be enclosed in brackets. Thus you might type PRINT PJOB (mandelbrot). From within a program you might wish to double a jobs priority:

```
150 PRINT 'Do you wish to speed up mandelbrot?'
```

```
160 answer$ = INKEY$ (-1)
170 IF answer = 'Y' THEN
180 priority = PJOB (-1)
190 priority = 2 * priority
200 SPJOB -1,priority
210 END IF
```

If mandelbrot is a SuperBasic program the inclusion of a section similar to the above would allow you to speed up the job. Of course, this only has an effect if other jobs are running, if only one job is active on the computer it will take the same amount of time to run, if its priority is 1 or 127.

OJOB returns the id of a jobs owner (ie. the job from which it is activated). In our example, PRINT OJOB (mandelbrot) would print 0, the id of SuperBasic.

JOB$ returns the name of a job given its id. Thus, PRINT JOB$ (1, 0) would print 'mandelbrot'. This is useful in any programs that refer to other jobs: you may wish to job ids in calculations but when it comes to displaying information it is better to convert to the job name.

NXJOB returns the id of the next job in the job tree. In our example NXTOB (0) would have a value of 1 ie. the next job after SuperBasic is mandelbrot. As more jobs are activated on the machine the job tree becomes more complex. Jobs may be activated by SuperBasic or by another job.

Super Toolkit II thus provides a set of commands and functions for controlling jobs and finding out information about jobs. It would be nice to have more functions for example, a function that returned the location of a job in memory, its length, the location and length of data it is using etc., etc. Much of this information can be found using machine code.

Preface

The original QL Toolkit was produced in something of a rush to provide
useful facilities which, arguably, should have been built in to the QL
to start with. Since its appearance, I have been subjected to
continuous pressure to modify certain facilities and extend the range
of facilities provided.
QLToolkit II is, therefore, a revised (to the extent of being almost
completely rewritten) and much enlarged version of the original 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 QLToolkit II ever saw the light of day is due to
prompting from a number of quarters. Many people have contacted me
complaining that they have been unable to lay their hands on the
original QLToolkit, and this eventually convinced me that there was a
market for a second version. Repeated criticism of the original
facilities made at great length (and with justification) by Chas
Dillon have provided the basis for many of the modifications to the
old routines. Ed Bruley has provided invaluable practical support in
putting the product on the market, and Cambridge Systems Technology
allowed me to use one of their Winchester disk 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! Bother, 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

SuperBASIC is used as a command language on the QL as well as a
programming language. Extensions are provided to improve the
facilities of SuperBASIC in both these areas as well as providing
program development facilities.
The following list gives a comprehensive form of each command or
function. There are often default values of the parameters to simplify

the use of the procedures.

2.1 Development Facilities

Section 3 File editing

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

Commands

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

2.2 Command Language

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

Section 4 Directory Control

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

Commands

DATA_USE name set the default directory
for data files
PROG_USE name set the default directory
for executable programs
DEST_USE name set the default destination
directory (COPY, WCOPY)
SPL_USE name set the default destination
device (SPL)
DDOWN name move to a sub-directory
DUP move up through the tree
DNEXT name move to another directory
at the same level
DLIST #channel 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, size, data as SEXEC but overwrites
file if it exists

Section 8 Program Execution

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

Commands

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

Section 9 Job Control

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

Commands

JOBS #channel list current jobs
RJOB id or name, error code remove a job
SPJOB id or name, priority set job priority
AJOB id or name, priority activate a job

Functions

PJOB (id or name) find priority of job
OJOB (id or name) find owner of job
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 onto a file
GET #channel\position, items get internal format data
from a file
PUT #channel\position, items put internal format data
onto a file
TRUNCATE #channel\position truncate file
FLUSH #channel flush file buffers
Functions
FPOS (#channel) find file position
Section 13 Format Conversions
Toolkit II provides a number of facilities for fixed format I/O.
These include binary and hexadecimal conversions as well as fixed
format decimal.
Commands
PRINT_USING #channel, format, fixed format output
list of items to print
Functions
FDEC$ (value, field, ndp) fixed format decimal
IDEC$ (value, field, ndp) scaled fixed format
CDEC$ (value, field, ndp) decimal
FEXP$ (value, field, ndp) fixed exponent format
HEX$ (value, number of bits) convert to hexadecimal
BIN$ (value, number of bits) convert to binary
HEX (hexadecimal string) hexadecimal to value
BIN (binary string) binary to value
Section 14 Display Control
Toolkit II provides commands for enabling and disabling the cursor
as well as setting the character fount 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 fount

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 occurred
REPORT #channel, error number report an error
CONTINUE line number continue or retry from a
RETRY line number 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'
keystrokes received on this channel.

Commands

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

Section 21 Microdrive Driver

Toolkit II extends the microdrive driver to provide OPEN file with
overwrite, as well as TRUNCATE and RENAME of files. These
facilities are supported at QDOS level (Traps #2 and #3) as
well as from SuperBASIC. The FLUSH operation is 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 file
server name
Device names
Nstation number_IO device the name of a remote
IO device (e.g. N2_FLP1_
is floppy 1 on 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 'CTRL F4' (press CTRL 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 arrow move cursor down a line
ALT up arrow scroll up a line (the screen moves down!)
ALT down arrow scroll down a line (the screen moves up!)
SHIFT up arrow scroll up one page
SHIFT down arrow scroll down one page
left arrow move cursor left one character
right arrow move cursor right one character
CTRL left arrow delete 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).

```
MDV2_
____________I___________
I I
JOHN_ OLD_
________I__________
I I
DATA1 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_.

```
MDV2_
________I______
I
JOHN_
____________________I_________________
I
TEST_
___________________I__________________
I I I I I I
ASM LIST REL LINK MAP BIN
```

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 progam 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 all three defaults:
DLIST list data, program and destination
or DLIST #channel defaults
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 #1
or WDIR #channel	list current directory to #channel
or WDIR \name	list current directory to 'name'
or WDIR name	list directory 'name' to #1
or WDIR #channel, name	list directory 'name' to #channel
or WDIR \name1, name2	list directory 'name2' to 'name1'

E.g.

WDIR \ser, _asm	list all _asm files in current directory to SER
WDIR flp1_	list all files on FLP1_ in window #1
WDIR #3	list all files in current directory to channel #3

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

5.3 Drive Statistics

There is one command to print the statistics for the drive holding a specified directory, or the data default directory.
STAT #channel, name
or STAT \name1, name2
Both the channel and the name are optional.

5.4 File Deletion

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

DELETE name	delete one file
WDEL #channel, name	delete files

For WDEL both the channel and the name are optional.
E.g.

WDEL	delete files from current directory
WDEL _list	delete all _list files from current directory

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

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

5.5 File Copying

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

Files in QDOS have headers which provide useful information about the file that follows. It depends on the circumstances whether it is a good idea to copy the header of 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 separator TO is used for clarity, you may use a comma instead.
If the channel is not given (i.e. most of the time), then the requests for confirmation will be sent to the command channel #0. 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 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 channel3 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 Programs

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

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

6.1 DO

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

DO name	do commands in 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 (>32 kbytes) program

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

For SEXEC and SEXEC_O the 'data' parameter is the default data space required by the program.
If there are any Jobs in the QL (apart from Job 0 the SuperBASIC interpreter) then LRESPR will fail with the error message 'not complete'. If this happens, use RJOB to remove all the other Jobs.

8 Program Execution

There is one procedure 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 D0 set to the error code.
E.g. RJOB 3,8,-1 remove Job 3, tag 8 with error -1
SPJOB demon,1 set the priority of the Job called
'demon' to 1
9.2 Job Status Functions
The Job status functions are provided to enable a SuperBASIC program
to scan the Job tree and carry out complex Job control procedures.
PJOB (id or name) find priority of Job
OJOB (id or name) find owner of Job
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 0, otherwise it will return the id of the next Job in
the tree.
Job 0 always exists and owns directly or indirectly all other Jobs in
the QL. Thus a scan starting with id = 0 and top Job id = 0 will scan
all Jobs in the QL.
It is possible that, during a scan of the tree, a Job may terminate.
As a precaution against this happening, the Job status functions
return the following values if called with an invalid Job id:
PJOB=0 OJOB=0 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:
outch = FOP_NEW (fred) :REMark open fred
if outch < 0: REPORT outch: STOP :REMark ... oops
PRINT #outch, 'This is file Fred'
CLOSE #outch
there is no need to ever know the actual channel number.
10.4 CLOSE
The CLOSE command has been extended to take multiple parameters. In
addition, if called with no parameters it will close all channel
numbers #3 and above. It will not report an error if a channel is not
open.
CLOSE #channels close channels
E.g. CLOSE #3, #4, #7 close #3, #4 and #7
11 File Information
There are six functions to extract information from the header of 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 is 0 for ordinary files
1 for executable programs
2 for relocatable machine code
The file information functions can also be used with implicit
channels. E.g.
PRINT FLEN (#3) print the length of the
file open on channel #3
PRINT FLEN (\fred) print the length of file
fred
12 Direct Access Files
In QDOS, files appear as a continuous stream of bytes. On directory
devices (Microdrives, hard disks etc.) the file pointer can be set to
any position in a file. This provides 'direct access' to any data
stored in the file. Access implies both read access and, if the file
is not open for read only (OPEN_IN from SuperBASIC, 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, items	get bytes from a file
BPUT #channel\position, items	put bytes onto a file

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

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

will put the byte values 4, 12 and 243 after the current file position
on the file open on #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 disk 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) hexadecimal to value
BIN (binary string) binary to value

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

14 Display Control

There are three separate facilities provided to extend the display control operations of the QL. They are cursor control, character 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 appropriate pattern from the second fount, failing that, it will use the first defined pattern in the second fount.

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

The format of a QL fount is:

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

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

The character 'g' is stored as: %00000000

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

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

CHAR_USE #channel, addr1, addr2 addr1 and addr2 both point
to substitute founts
CHAR_USE #channel, 0, addr2 the built in first fount
will be used, addr2 points
to a substitute second
fount
CHAR_USE 0,0 reset both founts for
window #1
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 occurred
ERNUM returns the error number
ERROR reporting
REPORT #channel reports the last error
REPORT reports the last error to
channel #0
REPORT #channel, error number reports the error number
given
RETRY and CONTINUE
As the RETRY and CONTINUE exit from an error clause without
resetting the WHEN ERROR, 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:

NETO_station number	output to station number
NETO_0	send broadcast
NETI_station number	input from station number
NETI_my station nunber	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 from
directory 'progs' on
Winchester disk 1 on
network station 1
SPL_USE n1_ser set the default spooler
destination to SER1 on
network station 1
It is possible to hide the network from applications by setting a
special name for 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 TRUMP CARD / GOLD CARD etc. 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 particularly 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$ = 1 : PRINT 'Fred wants coffee'
CLS #ch: CLOSE #ch
```

23 Writing programs to use with EX

Programs invoked by EX (or EW or ET) fall into three classifications:
non standard program header is not standard format;
special program header is standard but there
is an additional flag;
standard program header is standard.
So far as EX is concerned, the distinction is that a special program
must contain the code to open its own 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 jobs stack and the input pipe (if it is
required) opened and its ID put on the job's stack, then EX will make
a call to the address after the second identifying word. Note that the
code called will form part of a BASIC procedure, not part of an
executable program.

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

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

*these are the standard BASIC procedure 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 -nolis
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 wait for 530us
duration < 530us, if
contention occurs:
restart

b) header

4) hactiv set net active 22us wait for active
5) hbytes for each byte 11.2us for each byte wait for
start (inactive) bit, start (inactive) bit,
8*11.2us data bits, read 8 data bits, if
5*11.2us stop (active) fails: restart
bits
6) hackw wait for 2.5ms for set net active 22us
active, if not active:
restart
7) hackbt wait for start bit, send 11.2us start bit
read 8 data bits, 8 data bits 00000001
if error: restart

c) data

8) dactiv set net active 22us wait for active
9) dbytes for each byte 11.2us for each byte wait for
start (inactive) bit, start (inactive) bit,
8*11.2us data bits, read 8 data bits, if
5*11.2us stop (active) fails: restart
bits
10) dackw wait for 2.5ms for set net active 22us
active, if not active:
restart
11) dackbt wait for start bit, send 11.2us start bit
read 8 data bits, 8 data bits 00000001
if error: restart

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 wait for 530us
duration < 530us, if
contention occurs:
restart
4) scext send a scout extension
of 5ms active
b) header
5) hbytes for each byte 11.2us for each byte wait for
start (inactive) bit, start (inactive) bit,
8*11.2us data bits, read 8 data bits, if
5*11.2us stop (active) fails: nack
bits
6) hwait leaving net active,
wait 1ms
c) data
7) dbytes for each byte 11.2us for each byte wait for
start (inactive) bit, start (inactive) bit,
8*11.2us data bits, read 8 data bits if
5*11.2us stop (active) fails: nack
bits
8) dack inactivate net and within 500us set net
wait 1ms for active: active and wait 5ms,
if fails, restart do any processing
required and when ready
for next packet,
inactivate and restart
d) Not acknowledge
9) nack wait for inactive wait for 2.8us of active
or inactive, if inactive:
restart
10) nackw wait 500us for active: wait 200us for active, if
timeout is ok, active active: restart, if inactive
is fail 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
Directory 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 conversions (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	approx 1400
Buffered printer extension	approx 500
total	approx 2400

These can be accommodated 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 0, 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 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 asynchonous 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 I/O buffer damage when
interleaving serial I/O 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
and b) the file was not closed or flushed in this period
and 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 reorganised.
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 10 us. (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 error.

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 incorporated where appropriate.
V1.18 Verify introduced on restore; additional pauses introduced on seek error recovery.
V1.19 to V1.25 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
FXTRA 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
NFS_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

Configurable SuperToolkit II
This is the RAM based version of the SuperToolkit II. It is supplied
in a configurable form so that parts of the Toolkit may be left out if
they are not required. The full Toolkit occupies about 16k bytes, a minimum
useful Toolkit occupies about 1k bytes.
Before you start, have a formatted cartridge or disk ready for the configured
Toolkit.
To configure a Toolkit, put the cartridge or disk in drive one and reset the QL.
The configuration program will write a menu of 32 groups of Toolkit extensions.
These are grouped in the same order as the sections in the Toolkit manual.
The window at the bottom of the screen contains a description of the extensions
in each group.
When you have selected the extensions required, press F3 for a command menu.
Selecting 'Default directories' will produce prompts for the default directories
(see section 4). The default destination may be set to a device (e.g. SER1C) or
a directory (e.g. FLP1_BACK_) a default destination directory name must end with
an underscore. The program or data defaults are always directories.
When you 'make' the Toolkit you will be asked for a 'bootstrap' file name. This
is the file which will contain the SuperBASIC commands to load the Toolkit.
Put a formatted cartridge in the appropriate drive and give the full file name
of the bootstrap file (e.g. MDV1_BOOT). The configure program will write two
files to the medium, choosing the name of the second file for itself (it adds
_REXT to the bootstrap file name). You may configure more than one Toolkit at
a time, provided only that you specify a different bootstrap file name for each
one, or you change the medium.
To leave the configuration program press F3 then Q.
You may EXEC_W the configuration program (TK2_CONFIG) at any time. The QL does
not need to be reset.
As there are more than a thousand million combinations of facilities you can
choose, we have not been able to test all possible combinations. It is
possible that you may get a message 'configuration failed' or the resultant
Toolkit may malfunction. Please let us know if this happens.
Caution: To use the network extensions successfully you need a full speed
expansion RAM. (The only one known to us is the Technology Research Delta
Disk.)