QDOS / SMS

Reference Manual

0.0 Why this book?

First of all, many people asked for documentation about QDOS. The QL Technical Guide is out of print for some years, and it is impossible to get. The information is not up-to-date, and many things are missing. The Thing System documentation and the HOTKEY System II won't be modified too much in the future, so it makes sense now to explain how to use it. So that's why I thought it could be useful to make a new 'Operating System Guide'.

It took weeks to get this text typed in, and it took even more time to format everything, update the keys and text, and make sure that the text is as bug-free as possible. There will be typing-errors in the text, I'm sure, and if you find any serious mistake, please write. But, please make sure it is not a problem of your way of machine-code programming (QMON is quite helpful!). If you have serious questions and you cannot find an answer, please do NOT write, just call! If you really discovered a typing-bug, then you can write to

Jochen Merz Software	Tel. 0203/502011
Im stillen Winkel 12	Fax 0203/502012
47169 Duisburg	Mailbox 0203/502013
Germany	

Also, if you have written a useful application pointer-program of larger size and use, and you would like to see it distributed, then please send a copy of it to us. If it is a kind of program which is really worth marketing and selling, we could probably do it.

I take the chance and write some lines for those people who always find fault with the price, so I'm telling the story about QPTR: It was not half as hard to get the QPTR manual in a printable form; the text files from QJUMP were in ASCII-format with control codes embedded. Still, it took many, many days to get it converted into Text87 format, updated and printed. The update price (including a new 160 page manual with binder) is £13.50 (less than just a disc-update price of most other suppliers of computer software!), which leaves me about £6 after the costs for the printing, binder etc. are subtracted. Okay, there are some new customers of the product, but most orders are updates, and on the other side, there are advertising costs etc. If I double the number of currently sold QPTRs and updates, and count that against the hours used for producing the product, then this will result in less than 40 Pence per hour. Who would work for this? And, this does not consider the time taken to produce the individual copy, just the master. The question, why in the world do I spend my time, if it's not worth at all, is easy to answer: somebody has to do it, because this documentation is the basic for every pointer-program, and we urgently need new programs for the QL!!! This is also the reason for producing this book you are just reading: it is important to know how to program the QL, to keep it staying alive!

Back to this book: it is a mixture of the Technical Guide, The HOTKEY System II, the THING system, together with information about Level 2 device driver found in different hardware add-ons for the QL and the QL-Emulator for the ATARI ST, as well as some information about the QDOS-compatible operating systems SMS2 and SMSQ, and even more.

The keys used in this book are SMS-notation, as these keys are more meaningful then the keys used in the QL Technical Guide. You will also find these keys in the QPTR package. They have been introduced a few years ago, so it not only helpful but consistent. I decided not to put the old keys in brackets, as it is more confusing than helpful. People using the old keys will have the documentation; they probably do not need this book. People starting new projects should use the new keys, and if they use the Pointer Environment, they have to do so anyway.

This manual describes features available on all machines where not told otherwise. It assumes JS or MG or later ROMS. You may find some abbreviations in square brackets throughout the manual, they tell about restrictions. In general, try to program your programs that they don't collide with these restrictions. Where necessary, check software version and/or hardware to trap crashes.

[QL]	Only supported on QL, not on the QL-Emulator. This usually applies to hardware features, especially microdrives or the direct programming of the serial ports. These features may work on an emulator, but are not guaranteed.
[ST]	Only supported on the QL-Emulator for the ATARI-ST. This usually applies to hardware which does not exist on a QL. Will also work under SMS2 if it is running on an ST.
[SMS]	Needs the operating system SMS2 or SMSQ to be installed. Many features marked with [SMS] will also work on Qdos running on a QL-Emulator, but this is not guaranteed.
[SMSQ]	Needs the operating system SMSQ or SMSQ/E to be installed, preferably in the most recent version.
[not SMS2]	This feature is not supported on SMS2, so better avoid it if you want to write programs which run under all operating systems.
[DD2]	Only supported on Level 2 Directory Device Drivers. This depends on the hardware connected to your machine. Microdrives and old Floppy Disc drivers are not Level 2, whereas the Drivers for the Miracle Winchester (for example), or the RAM disc, Floppy Disk and Hard-Disk on the ST-Emulator (from Level C onwards) are Level 2. Devices on SMS are minimum Level 2.
[DV3]	Only supported on Level 3 Directory Device Drivers.
[EXT]	needs some kind of extension to be installed. This could be the HOTKEY System II, The Pointer Environment, or SuperToolkit II, for example. It could also be built into a hardware expansion, e.g. Floppy-Disc-Controller. In general: available for 'well equipped' users, especially QL-Emulator owners. Will be available in SMS2.
[QDOS Vx.xx+]	only supported from operating system versions x.xx onwards supported. Can have unpredictable results on older versions.

Credits: Many thanks to Tony Tebby for his permission to use a lot of his documentation for this book. Thanks also to a very helpful friend who checked the typing. Many thanks to all of those users who keep on asking for documentation - they showed interest which made me think of doing this book.

1.0 About this Guide

This guide describes the methods which may be used for machine-code programming on the QL. Its contents are also relevant to compiler writers who must implement a run-time library for other languages. This guide describes only those techniques which are specific to the QL. It does not contain a general description of 68000 or 68008 assembly language programming: this information can be obtained from a number of different sources. It is therefore, **strongly recommended that a reference book describing 68000 assembly language** be consulted before attempting to understand this guide.

The guide also gives details of how various peripherals such as hard disk interfaces, add-on memory and ROM cartridges may be added on to the QL, with many details about how the firmware for such devices should be written.

Readers may notice that there are no circuit diagrams or detailed explanations of the QL's internal hardware structure in this manual. This is because it is not necessary to have such information in order to write software for the QL. Sinclair tried in the design of Qdos to provide you with a stable interface to the machine through its operating system; everything you need is there and so long as you build your products using the interface provided there is no danger that any future upgrade of the QL will introduce an incompatibility with existing software products. Programs using supported entries only will work fine on future versions of the operating system, as well as on different hardware like the ATARI ST QL-Emulator or QXL card.

2.0 Introduction to QDOS / SMS

QDOS is the QL operating system. SMS is an advanced version, completely reprogrammed but as compatible as possible. It is a single-user multi-tasking operating system: that is, it provides the means for several independent programs to run concurrently in the QL, but does not provide any mechanisms to prevent those programs from interfering with each other. Qdos can be thought of as a collection of several things:

1. A set of useful routines for performing functions such as memory allocation, Input/Output, etc.

2. A mechanism for maintaining lists of things to be done on interrupt, including the function of allocating slots of CPU time to programs which require them.

3. A mechanism for starting up the computer, and determining the configuration of any add-on hardware that is connected to it.

The QDOS mechanisms for start- up are described in section 2.4. Once start-up has been performed, QDOS does not "run" in the sense that traditional operating systems run: its pieces of code and data structures simply exist for programs to use. There is no QDOS "main program" that maintains continuous control of the machine: the SuperBASIC interpreter, which takes the place of the command line interpreter found in traditional operating systems, is simply a program which runs on the QL and uses QDOS's facilities, albeit with a number of special provisions. It is possible, and indeed commonly done, to destroy the SuperBASIC interpreter completely, and yet still use all the facilities of the operating system.

Note that in this guide, hex numbers are preceded by a dollar sign ($) as used in the Motorola assembly language format.

2.1 Memory Map

This section describes how Qdos maintains its RAM area. On the standard QL, the RAM starts with the screen RAM at address $20000, and the area available to Qdos starts at $28000. In an unexpanded QL, the RAM finishes at $3FFFF, whilst in a QL with expansion memory, the RAM may go up as far as $BFFFF. The Qdos initialisation routine determines the amount of RAM present and adjusts the position of its pointers accordingly.

In an ST, RAM may end up at $3FFFFF. The current version of QDOS supports only a maximum RAM size of 4MB, so it can't be expanded any further.

The memory map is as follows:

SYS_ RAMT		
SYS_RPAB	Resident procedure area	
SYS_TPAB	Transient program area	
SYS_SBAB	SuperBASIC area	
SYS_FSBB	Free memory area (used up for slave blocks by the filing system)	
SYS_CHPB	Common heap area	
	System management tables	
	System variables	Base of system variables
	Display RAM	Base of RAM

2.1.1 Principles
There is no memory management hardware in the QL. This means that all code must execute from fixed addresses in physical memory, and that a piece of code may not be moved after it has been loaded into memory. For this reason, memory is usually allocated in fixed size areas which remain in a fixed location until deleted. The SuperBASIC area is an important exception to this.

2.1.2 System Variables
The QDOS system variables are a block of memory containing information required by the operating system.

This block is normally located at address $28000, but is not fixed at this address in principle. Applications programs should not rely on that fixed address, but should get the address of the base of system variables by calling the **SMS.INFO** trap (see section 13.0).

Some of the system variables can usefully be monitored by applications programs, and some of them can safely be altered. A complete list of the system variables, each with its size and offset from the base of system variables, given in section 18.2.

Included in the system variables area are a set of longword pointers indicating the locations of the other areas in the memory map.

2.1.3 System Management Tables

Immediately above the system variables are various tables used by QDOS to maintain the list of jobs and various other pieces of information. The supervisor stack also resides in this area.

2.1.4 Common Heap Area

The Common heap area contains the channel definitions which are maintained by the I/O sub-system, together with the working storage required by I/O drivers or programs. The allocation of space in this area is carried out either by device drivers, when invoked, or directly by jobs. There are two traps provided to allocate and release space in this area: **SMS.ACHP** and **SMS.RCHP** (see Section 13.0). The heap allocations of a job are automatically released when the job is removed.

The common heap is an example of the use of a general heap mechanism provided by QDOS, which operates in the way described in the entry for **SMS.ALHP** in section 13.0.

The user code needs to retain one pointer to the free space in the heap. This is a long word and is a relative pointer to the free space in the heap. When the heap has no free space, either because it does not exist, or because it is full, this pointer is zero.

2.1.5 Free Memory Area

The free memory area is used by QDOS as a buffer memory for the Microdrives, or, if QDOS is suitably extended, for other filing system devices. The area is structured as a collection of slave blocks, that is, blocks which are associated with a physical block on medium. When memory is allocated in another area which would encroach on the free memory area, QDOS must remove one or more slave blocks. Before such a removal takes place, QDOS ensures that a true copy of the information is present on the medium.

Whilst the common heap grows upwards into the free memory area, the areas above it grow downwards into it. As there are three areas above it (the resident procedure area, the transient program area and the SuperBASIC area), special provisions are made so that all three can grow at the appropriate times.

2.1.6 SuperBASIC area

The SuperBASIC interpreter owns a special area located immediately above the free memory area: this area is used for all the interpreter's storage requirements such as the SuperBASIC programs, its variables, its table of I/O channels and the interpreter's working storage. This area is noteworthy in that it can be moved by QDOS without the knowledge of the SuperBASIC interpreter if an area above it needs to grow, or if the SuperBASIC area itself needs to shrink. Its size may also be altered. The mechanism which makes such movement or alteration in size possible operates as follows:

All references to the SuperBASIC area are made relative to the address register A6, and the value of A6 on entry to the interpreter is adjusted by QDOS to the current base of the Super-BASIC area (which is held in the system variable **SYS_SBAB**), offset by the length of the interpreter's job header (currently $68 bytes).

The SuperBASIC interpreter divides its working area into several portions, details of which may be found by looking at the **BV** definitions in section 18.3. All of the pointers to these various portions are also relative to A6.

2.1.7 Transient Program Area

The transient program area is the area of memory into which the user's applications programs are loaded. Each job is allocated a block of memory in the transient program area, which it keeps until it is deleted: this area is used for the job's code, data and stack. Programs loaded in this way are not normally re-entrant, but it is relatively straightforward to use the mechanisms in the system to set up a single piece of code which is shared by several different jobs with different data areas.

There is no safe way of determining a priori where a program will be loaded, therefore programs are normally position independent (see section 3.1 on jobs).

2.1.8 Resident Procedure Area

Memory allocated in this area is unavailable to the operating system. The system knows only two things about the resident procedure area: how to allocate memory in it, and how to release it completely. Both of these operations can only be carried out when there are no transient programs in the machine, due to the fact that the transient program area cannot be moved. Normally, the allocation is done immediately after start-up, and deallocation is never performed.

The area is normally used to load in machine code procedures and functions written to extend the SuperBASIC language (see section 9.7), and occasionally for loading in the code of device drivers when these are not located in ROM in an add-on device.

2.2 Calling QDOS/SMS Routines

There are two categories of QDOS routines available to the user: traps and vectored routines. The mechanism for calling a routine is different for each of these two categories.

2.2.1 Traps

Traps are called using the 68008 TRAP #n instruction: on the QL, this has the effect of a subroutine call to a defined location which has the side effect of saving the status register and entering supervisor mode.

Of the sixteen trap numbers available on the 68008, numbers 0 to 4 inclusive are defined for use by QDOS, the remainder being free for the user to redirect to his own routines. Roughly speaking, the traps are utilised as follows:

TRAP #0	Special trap for entering supervisor mode.
TRAP #1	Manager traps - routines which perform overall control of the hardware and of the operating system's resources.
TRAP #2	Input/ Output management traps (I/O traps which allocate resources).
TRAP #3	Input/ Output traps which do not allocate resources.
TRAP #4	Special trap for the SuperBASIC interpreter.

Traps are called by setting up any required parameters in registers A0-A3 and D1-D3, setting up the code for the required trap in D0 (usually with a MOVEQ instruction), then executing the TRAP instruction. Trap routines do not affect D4 to D7 or A4 to A6. There are, however, a few defined cases which are exceptions to this.

When the TRAP operating is complete, control is returned to the program at the location following the TRAP instruction, with an error key in all 32 bits of D0. This key is set to zero if the operation has been completed successfully, and is set to a negative number for any of the system-defined errors (see section 17.1 for a list of the meanings of the possible error codes). The key may also be set to a positive number, in which case that number is a pointer to an error string, relative to address $8000. The string is in the usual QDOS form of a word giving the length of the string, followed by the characters.

Note that all traps can return the error code **ERR.IPAR** (for invalid parameter). Note also that the condition codes may not be set according to the error code on return from a trap, thus a program wishing to detect an error should execute a TST.L D0 instruction immediately after the TRAP instruction.

Details of all the QDOS traps are given in sections 13.0-15.0.

2.2.2 Vectored Routines

In addition to the routines accessed by traps, there are several utility routines which are available to the applications program: their addresses are held in a vector table which is located in the ROM starting at address $C0. A vectored routine can be accessed by the following code:

```
MOVE.W    VECTOR_ADDRESS,An
JSR       (An)
```

where **VECTOR_ADDRESS** is the address of the vector table entry, and An is a suitable address register which is not required by the particular routine on entry.

There are some exceptions to this technique: for some vectored routines, the code is:

```
MOVE.W    VECTOR_ADDRESS,An
JSR       $4000(An)
```

The entries in section 16.0 for vectored routines which require this treatment are suitably marked.

There are no general rules covering the handling of errors in vectored routines. Some routines return an error code in D0 in the same way as traps, but others use the technique of returning to one of a set of alternative return addresses. An example is the vectored routine **MD.RDHDR**, which returns to the location after the call if there is a "bad medium" error detected, to the address 2 bytes later if there is a "bad sector header" error detected, and to the address 4 bytes later for a correct completion. Thus the correct code to trap these errors would be:

```
MOVE.W    VECTOR_ADDRESS,An
JSR       $4000(An)
BRA.S     BAD_MEDIUM_ERROR
BRA.S     BAD_SECTOR_ERROR
```

* Code for processing a correct return starts here
"

```
BAD_MEDIUM_ERROR
```
* Code for processing a bad medium error starts here
"

```
BAD_SECTOR_ERROR
```
* Code for processing a bad sector error starts here

Obviously, a similar mechanism can be used with any number of error returns (including zero or one).

Complete details of the vectored routines are given in section 16.0, including information about the behaviour of each routine when an error occurs.

2.2.3 Atomic Actions

In general, system calls are treated as atomic: while one job is in supervisor mode, no other job in the system can take over the processor. This provides for resource table protection without the need for complex procedures using semaphores. If a job needs to execute some action other than a single system call into which no other job must be allowed to intervene, it should enter supervisor mode before entering the code which performs this action. Supervisor mode is entered using **TRAP #0**. The stack pointer only is changed by this trap.

A job should only use 64 bytes on the supervisor stack, and all of the space used on this stack **must** be released before exiting supervisor mode. In general, there should be nothing on the supervisor stack when a manager trap is made.

Some system calls are only partially atomic, that is, when they have completed their primary function, some other job may gain a share of CPU time before control returns to the calling job. These partially atomic system calls must not be made from a job in supervisor mode. All of the scheduler calls (i.e., **TRAP #1** with D0 = 4, 5, 8, 9, $A, $B) fall into this category, as do all the I/O calls (**TRAP #3**), unless immediate return (timeout=O) is specified.

A piece of code in supervisor mode can be interrupted by the frame (50/60 Hz) or external interrupts, so care must be taken, when writing interrupt servers, that the system' s internal data structure is not modified, directly or indirectly, by system calls. In practice, since interrupt servers tend only to be moving data into or out of queues, this is not a serious limitation.

2.3 Exception Processing

There are three categories of exception traps on the 68008: user traps, traps for software error conditions, and traps for hardware interrupts. There is also one special hardware trap called "bus error", which can be used to trap bad conditions on the address bus: this trap is not supported by the QL hardware.

User traps 0 to 4 inclusive are treated as defined in sections 13.0 through 15.0.

User traps 5 to 15 inclusive, together with the software error traps for "address error", "illegal instruction", "divide by zero", "check array", "trap on overflow", "privilege violation" and "trace" are redirectable by the user on a per-job basis: see the entry for **SMS.EXV** in section 13.0.

Traps and exception vectors which are not used by QDOS may be redirected through a table which is set up by particular job.

If a job has set up a table of trap vectors for itself, then that table will automatically be used when that particular job is being executed. The vector tables used by other jobs will not be affected. A job set up by, even if not owned by, a job which has set up a table of trap vectors, will use the same table as that job, until it is redefined.

If the job ID is a negative word, then the table will be set up for the calling job.

The table is in the form of a long word address for each trap or exception. They are in the following order:

$00 address error
$04 illegal instruction
$08 zero divide
$0C CHK
$10 TRAPV
$14 privilege violation
$18 trace
$1C interrupt level 7
$20 trap #5
$24 trap #6
$28 trap #7
$2C trap #8
$30 trap #9
$34 trap #10
$38 trap #11
$3C trap #12
$40 trap #13
$44 trap #14
$48 trap #15
$4C end of table

All interrupts on the QL are auto-vectored, therefore there is no treatment of the 68008 vectored interrupt traps. Interrupts generated by the QL internally are level 2 auto-vectors: the interrupt handling mechanism includes the facility for detecting an interrupt on the EXTINTL (external interrupt, active low) line in the QL's expansion port.

It is also possible to generate a level 7 (non-maskable) interrupt: the treatment of this can also be redirected on a per-job basis. Pressing CTRL-ALT-7 on the keyboard generates a level interrupt and also resets all communications with the IPC: a suitable interrupt handler could be written to perform a warm start on the system to allow partial recovery from a crash.

2.4 Start-up

The first thing that QDOS does when the system is reset is to execute a RAM test. This test determines the amount of contiguous RAM present, and if there is any RAM failure, hangs up the machine.

QDOS then initialises the system variables, the system management tables, and the Super-BASIC area.

The address **$C000** is then checked by QDOS for the characteristic longword **$4AFB0001**: if this is found, QDOS links in the SuperBASIC procedures contained in the ROM, prints out the name of the ROM, and performs a **JSR** to its initialisation point (details of the correct format of the **ROM** are found in section 11.4). It is perfectly in order for the code in this ROM to take over the machine completely and never return to the system, for example if another operating system were being booted.

QDOS then does the same for the other ROMs in the expansion slots.

If all of these ROMs return control to QDOS, the next action is to try to open a device driver "BOOT": if this is found, its contents are loaded as a SuperBASIC program and executed. If no device driver "BOOT" has been linked in, QDOS attempts to find a file "MDV1_BOOT" and load and execute its contents as a SuperBASIC program. If both of these attempts fail, QDOS starts up the SuperBASIC interpreter with an empty program memory.

3.0 Machine Code Programming

Four types of machine code are available to program the Ql, each being used to perform quite different operations: jobs, SuperBASIC procedures and functions, tasks, and the operating system or extensions to it. Thus there are several differences in both the form in which they are written, and the way in which they are treated by Qdos.

3.1 Jobs

Most application programs written in machine code or compiled code will be in the form of jobs. A job is an entity which has a share of machine resources: it has a priority which allows it to claim time-slots of CPU activity, and it has a fixed-size area of memory where data and code can be stored: code normally starts at the bottom of the area, and data at the top. This area is located somewhere in the transient program area.

Note that the command interpreter is itself a job but with the exceptional characteristic that its data area is expandable.

A job also has the ability to **own** I/O channels or other jobs. There is no protection between jobs under Qdos, so that channels are available for use by all jobs. Ownership simply implies that when the owner of a channel or job is deleted, the owned channel or job is deleted also (this process continues recursively).

Jobs have three well-defined states: they are active, sharing CPU resources with other jobs; suspended, for example, waiting for I/O or another job; or inactive, occupying memory but not capable of using CPU resources.

The priority of a job can be zero, in which case it is suspended, and does not consume CPU time. It can in fact be suspended for its entire lifetime and never execute at all, which would be the case if it was simply used as a means of obtaining some memory into which data could be loaded. A job at any other priority level is active.

When a job is started, two parts of its area of memory have defined meanings: the bottom of the code area, and the stack, which is at the top of the data area. It is the programmer's responsibility to set up the bottom of the code area, which should be in the following form for use by Qdos utilities:

```
     JMP.L     JOB_START
     DC.W      $4AFB
     DC.W      JOB_NAME_LENGTH
     DC.B      'Name of job' (word-aligned)
JOB_START
* Code begins execution here (assuming that the
* start address defined when the job was created was zero)
```

On the first occasion that a job is activated, (A6) points to the base of the job area, (A6,A4) points to the bottom of the data space, and (A6,A5) points to the top of the jobs area. There may be some information on the stack, which will be in the following form: (A7) points to the number of channels which have been opened for the job before it was activated; above this is a sequence of longwords holding the channel IDs, and above these are a command string which may have been passed to the job. It is the programmer's responsibility when starting a job to set up this information: the SuperBASIC **EXEC**, **EXEC_W** commands and any utilities produced by Sinclair are compatible with this form.

```
(A6,A5)    | Command string        length(word) + bytes     |
           |                                                |
           | Channel ID            long                     |
           | Channel ID            long                     |
           | "                                              |
           | "                                              |
           | Channel ID            long                     |
           |                                                |
(A7)       | Number of Channel IDs word                     |
           |                                                |
(A6,A4)    | Data area                                      |
           |                                                |
           | Code area                                      |
           |                                                |
           | Job name              length(word) + bytes     |
           |                                                |
           | $4AFB                 word                     |
           |                                                |
(A6)       | JMP.L   JOB_START                              |
```

Note that the normal sequence in Qdos is as follows:

1. reserve space for a job;
2. load its code in;
3. open its channels;
4. activate it.

Execution begins at an address specified when the job was created. This is normally specified as zero, which is why the first thing in a job is normally a **JMP.L** instruction to the entrypoint of the code. Since Qdos cannot give guarantees as to where a job will be loaded, it is usual to write jobs as position-independent code, although it is possible to avoid this constraint if a special relocating loader is used after the space for the job has been allocated.

The system job table holds information about the jobs within the system. The system variable **SYS_JBTB** points to the base of the job table, and **SYS_JBTT** points to the top. The table is a series of longwords each of which points to a job control block: the contents of this are described in section 18.5. The job is identified to the system by its JOB ID: this is a longword consisting of a word giving its position in the job table (in the least significant word), and a word of tag allocated by the operating system when the job is created (in the most significant word).

The traps that may be called relating to jobs are as follows:

SMS.INFO	returns the current job ID, plus miscellaneous information
SMS.INJB	returns the status of a job
SMS.CRJB	creates a job
SMS.RMJB	removes an inactive job
SMS.FRJB	forces removal of a job (whether inactive or not)
SMS.FRTP	finds the largest space available for a job
SMS.EXV	sets the trap-vector table for a job
SMS.SSJB	suspends a job
SMS.USJB	releases a job
SMS.ACJB	activates a job
SMS.SPJB	changes the priority of a job

A job terminates itself by calling **SMS.FRJB** with its own job ID (or -1, which always refers to the current job).

3.2 SuperBASIC Procedures and Functions

The SuperBASIC command interpreter is job number zero. It behaves like all other jobs in most respects, with the important exception that it owns a special data area which is expandable, and may be moved without the knowledge of the interpreter. This area is located immediately below the transient program area.

Machine code procedures and functions which are added to SuperBASIC appear to the user to be identical to those which are built into the ROM. From the user's point of view they are routines which are executed from within job number zero, but which have certain constraints on the way they are coded.

The most important constraint is that A6 is used to point to the (moveable) base of the Super-BASIC data area. The system may move the area and change the value of A6 between instructions without the knowledge of the interpreter, therefore A6 must not be modified within the procedure or function, and its value must not be stored or used in calculation. This constraint may be side-stepped by entering supervisor mode, but A6 must then be restored on exit back to user mode (the processor is in user mode when a procedure or function is entered). The stackpointer A7 must of course be restored to its original value before exiting from the procedure.

On exit from the procedure, an error key is passed to the interpreter in D0.L: this must be set to zero if there was no error. The procedure or function can then be exited using an **RTS** statement.

If machine code procedures or functions are to be used either recursively or in recursive Super-BASIC procedures, they must obey the usual constraints of having no local variables and no self-modifying code.

Machine code procedures and functions are normally loaded into the resident procedure area above the transient program area. This area can only be expanded or deleted when the transient program area is empty, which is normally immediately after the machine is booted.

Trap #4 is the one special trap which relates to SuperBASIC procedures and functions. This trap is used to make the addresses passed to an I/O trap relative to A6, which is necessary when working with the SuperBASIC variables area. It only affects the following trap, and must therefore be called before each trap whose addresses are to be modified.

Details of parameter passing, function returns and other useful information about the Super-BASIC interface are given in section 9.0.

3.3 Tasks

Tasks are special pieces of code invoked under interrupt, usually as part of the physical layer of a device driver. They obey special rules according to the precise conditions under which they are called: these rules are described in the sections on device drivers (sections 6.0-8.0). The important restriction on tasks is that they must not allocate or release machine resources: this should only be done from within a job, or within the access layer of a device driver.

3.4 Operating System Extensions

Some parts of user-defined device drivers do not fit into any of the above categories: they are special routines called from within a job via the Qdos Input/ output sub-system (see section 6.0). These routines have their own rules, and these are described in the sections on device drivers (sections 6.0-8.0).

3.5 Special Programs

Special Programs have, like standard jobs, 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). This is followed by a further value of $4AFB (aligned on a word boundary). When the program has been loaded, the option string put on the jobs stack and the input pipe (if 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 call will form part of a Basic procedure, not part of an executable program.

Special Program

Call parameters		Return parameters	
D1-D3		D1-D3 ???	
D4.L	0 or 1 if there is an input pipe ID is not on stack	D4	???
D5.L	0 or 1 if there is an output pipe ID is on stack	D5	nr. of channel ID's on stack
D6.L	job-ID for this program	D6	???
D7.L	total nr. of pipes and filenames	D7	???
A0	address of support routines	A0	???
A1	pointer to command string	A1	???
A2		A2	???
A3	pointer to first filename (name table) (relative to A6) *	A3	???
A4	pointer to job's stack	A4	
A5	pointer beyond last filename (name tab.) (relative to A6) *	A5	???
A6	base pointer	A6	preserved

Error returns: any standard returns

The entries marked with * are relative to A6 (standard SuperBASIC procedure passing registers, see Section 9.8).

The file setup procedure should decode the filenames, open the files required and put the IDs on the stack (A4). D5 must be incremented by the number of channel IDs put on the job's stack.

The routine **(A0)** to get a filename 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 returned 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 filename in A1 (relative to A6). The filename should not be in the Basic buffer; D3 should hold the access code and the job ID (as passed to the initialisation code) should be in D6. The error code is returned in D0, while D1 and D2 are smashed, and A1 is returned pointing to the filename used (it may have a defualt 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.

4.0 Memory Allocation

Memory is allocated differently in each area of the Qdos memory map.

* Memory in the resident procedure area is allocated using the trap **SMS.ARPA.**

* Memory in the transient program area is allocated by the mechanisms described in section 13.0 for creation and deletion of jobs. The vectored routines **MEM.ALHP** and **MEM.REHP** may be used within a job to perform primitive heap allocation inside that job's own data area.

* Memory in the SuperBASIC area is allocated by various mechanisms. The traps **SMS.AMPA** and **SMS.RMPA** are used by the interpreter to change the size of the entire area, but are not normally used by anything else. The vectored routine **QA.RESRI** is used to allocate space on the arithmetic stack: the interpreter itself cleans up this space on return from a procedure or function. Space in the remaining parts of the SuperBASIC area is usually allocated by the vectored routines being used to perform the operations that require the space, so that this allocation is invisible to the user, except that it usually results in a modification of the value of A6.

* Memory in the free memory area is not allocated or deallocated by the user, except by the slave block mechanisms defined in section 7.0 on directory device drivers.

* Memory in the common heap is allocated and released by the traps **SMS.ACHP** and **SMS.RCHP**. The area allocated in this way by a job is released when that job is deleted. The same mechanisms can be accessed from within device drivers via the vectored routines **MEM.ACHP** and **MEM.RCHP**.

4.1 Heap Mechanism

The mechanism for allocating and releasing space are common to various routines. They are as follows:

A heap is an area of memory which contains a linked list of used heap items, and a linked list of free heap items. Each heap item is an area of memory (which is a multiple of 8 bytes long), together with a pair of longwords: the first is the length of the heap item, while the second is a pointer (relative to itself) to the next heap item in the list. The use of relative pointers ensures that heaps may be moved.

A heap is set up by linking an area of ram -> memory into a non-existent heap (free space pointer = 0). A heap is expanded by linking an area of ram -> memory, preferably but not necessarily, contiguous with the current top of the heap, into the heap.

Provided the user code can remember the length of a heap item, all of the memory in it may be used by the code. On allocation of the heap item, the first long word holds its length, and so, if desired, this may be retained by the user code.

The user code requires to keep one pointer to the first free space item in the heap. This is a long word, and is relative. When the heap has no free space, either because it does not exist, or because it is full, this pointer is zero.

Releasing a heap item adds it to the list of free space items within the heap, and consolidates it with adjacent free spaces where appropriate.

5.0 Input/ Output on the QL

A QL program uses I/O by accessing Qdos. The IOSS in turn accesses the device driver for the appropriate device. The device driver is a piece of code which can perform low- level I/O routines for a particular device: that device may correspond to a piece of hardware, such as a serial port, or it may be some notional device occupying a piece of memory, such as a pipe, which is a communication channel between jobs.

QL I/O is performed through the IOSS using an I/O channel. The applications program opens a channel by passing a device name to the IOSS, which returns a channel ID. The IOSS and the built-in device drivers have the ability to recognize qualifiers appended to the actual name of the device which can direct the open operation in particular ways, such as identifying a file name, or selecting some hardware option. The program then uses the channel ID to identify to the IOSS which channel it wishes to access when performing read or write operations on it. It can also close the channel, passing the channel ID to the IOSS. There may be several channels open which use the same device driver, such as multiple screen windows, or Microdrive files. For this reason, all the built-in drivers are re-entrant, as must user-defined drivers if they are to have the same capability.

The QL ROM contains drivers for several devices such as screen windows, serial ports. pipes, microdrives, and so on. The user can add his own device drivers for pieces of add- on hardware, or simply for additional functions with the existing hardware.

Note that a channel ID is not the same thing as a SuperBASIC channel number (denoted by # expression): the latter is the index of an entry in the SuperBASIC channel table which includes a channel ID. See sections 18.4 and 18.7 for details of the channel table.

5.1 Serial I/O

All device drivers have, at the very least, the capability to perform serial I/O: that is, the operations of reading bytes, writing bytes, and testing for pending input. Serial I/O is completely byte-oriented - unlike many operating systems there is no inbuilt record structure, which means that the user is free to superpose his own record maintenance in whatever form he wishes. I/O which is purely serial is completely redirectable: when different devices are being used, the device name passed to the channel open trap is the only thing that changes.

The IOSS supports one control character only, this being the newline character, which is ASCII 10 ($0A). Whilst this has the disadvantage that one cannot directly store files of graphics commands which can be retrieved by a simple copy, it does have the advantage that files containing arbitrary sequences of bytes cannot do irretrievable damage to the system by being copied to a device for which they were not intended. The serial driver has the option of supporting ASCII 13 as a newline, and ASCII 26 (CTRL-Z) as an end of file marker.

All serial I/O calls support a time-out feature, which may be zero (return immediately), indefinite (wait until the operation is complete), or finite (wait until the operation is complete, or for a set time, whichever is the sooner). This last feature makes it very easy to write code which, for example, puts up a menu only if the user hesitates.

The IOSS supports the following calls for serial I/O:

IOA.OPEN	opens a channel
IOA.CLOS	closes a channel
IOB.TEST	tests for pending input
IOB.FBYT	fetches a single byte
IOB.FLIN	fetches a line of bytes terminated by newline (ASCII 10)
IOB.FMUL	fetches a string of bytes
IOB.SBYT	sends a single byte
IOB.SMUL	sends a string of bytes

The fetch and send traps have several special meanings when used in conjunction with screen or console channels: for a more detailed description of these, see section 15.0 on I/O Traps.

For the fetch byte and fetch string traps, characters read from the keyboard are not echoed in the associated window, and cursor handling is left to the applications program.

5.2 File I/O

Qdos files appear to the applications program as arrays of bytes on a physical device, with an associated file pointer which gives the "current position" in a file. A file also has a header, which is normally 64 bytes long containing information about the file such as its name, length, etc. Further details concerning the format of the file header are given in section 7.0 on Directory Device Drivers.

The open call to a file system device supports several modes: old (exclusive), old (shared), or new (exclusive). New (overwrite) mode has a slot allocated in the open keys, but is not currently supported for Microdrives. In addition, a special open key indicates that it is desired to open the directory of the medium for reading rather than a particular file; the directory cannot be explicitly written, but is maintained by the device driver when open calls and deletions are made.

Qdos supports a system of slaving, whereby 512-byte blocks of data are buffered in the free memory area (see section 4.0): all unused memory being taken for this area. The filing system may return from a write operation when that operation has only been performed on the slave block concerned; Qdos will later force the system to convert that slave block into a true copy of the data on the physical device. As a result of this mechanism, add-on filing devices normally support 512-byte logical blocks: however this blocking system is transparent to the applications program. A single slave block table is shared by all the directory drivers which want to use it to improve their performance.

In addition to the serial I/O operations described above, Qdos supports the following operations for file-system devices:

IOA.FRMT	formats a sectored medium
IOA.DELF	deletes a file
IOF.CHEK	checks all pending operations on a file
IOF.FLSH	flushes buffers for a file
IOF.POSA	positions the file pointer absolutely
IOF.POSR	positions the file pointer relatively
IOF.MINF	gets information about the mounted medium
IOF.SHDR	sets the file header
IOF.RHDR	reads the file header
IOF.LOAD	loads a file into memory
IOF.SAVE	saves a file from memory

The **IOF.FLSH** and **IOF.TEST** commands are subtly different: **IOF.FLSH** ensures that all write operations are complete, whereas **IOF.TEST** ensures that all write and read operations (including prefetches) are complete.

5.3 Screen and Console I/O

The keyboard and screen devices are treated in a special way by Qdos, and have a large number of functions in addition to those available for purely serial I/O devices. Two types of device are supported: **scr** (for screen), which is a screen window, and **con** (for console), which is a screen window with an associated keyboard channel. The three channels #0, #1 and #2 which are opened by SuperBASIC are all console channels.

5.3.1 Display Modes

The QL has two display modes (see the **Concepts** manual for details). The display mode can be set or read using the **SMS.DMOD** trap, but as this trap clears all screen windows, it should be used with great care. A program can also find out whether the user selected TV or monitor at switch-on by inspecting the value of the system variable **SYS_DTYP, which is unfortunately smashed by the MODE command on standard QLs.**

There are two main coordinate systems used for screen I/O: these are the graphics coordinate system and the pixel coordinate system (see the **Concepts** manual for details). Note that in 256-pixel mode and for several commands in 512-pixel mode, the least significant bit of a dimension in the x-direction is ignored, so that a given pixel address refers to the same location in both modes. Some traps refer to character coordinates: these are based on the pixel coordinate system but are scaled by the current character spacing for the window.

5.3.2 Window Properties and Operations

A window is an area of screen which may be in any position on the screen, subject to the restriction that its x-position must be an even number. A window may be of any size that does not run off the edge or bottom of the screen, subject to the same restriction. Windows may overlap, but the system does not store or retrieve the area of overlap, it being the user's responsibility to ensure that any information is not lost or garbled.

Each window will have its own particular set of characteristics: a border width, a border colour, a paper colour, a strip colour, an ink colour, a cursor position, a cursor increment, a flag which says whether the cursor is suppressed, a pair of font pointers, information about newline treatment, and graphics information. Details of the window definition block are given in sections 18.7 to 18.10.

The special traps for dealing with windows are as follows:

IOW.PIXQ	returns window information in pixel coordinates
IOW.CHRQ	returns window information in character coordinates
IOW.DEFB	set the border width and colour
IOW.DEFW	redefines a window
IOW.ECUR	enables the cursor
IOW.DCUR	suppresses the cursor
IOW.SCRA	scrolls a whole window
IOW.SCRT	scrolls the top part of a window
IOW.SCRB	scrolls the bottom part of a window
IOW.PANA	pans a whole window
IOW.PANL	pans the line the cursor is on
IOW.PANR	pans the the right-hand end of the line the cursor is on
IOW.CLRA	clears a whole window
IOW.CLRT	clears the top part of a window
IOW.CLRB	clears the bottom part of a window
IOW.CLRL	clears the line the cursor is on
IOW.CLRR	clears the right-hand end of the line the cursor is on
IOW.RCLR	recolours a window

IOW.SPAP	set the paper colour
IOW.SSTR	set the strip colour
IOW.SINK	set the ink colour
IOW.BLOK	fills a rectangular block in a window
IOW.SOVA	set the character writing or plotting mode

5.3.3 Screen Character Output Operations

Newline characters receive slightly different treatment when bytes are being sent to a screen or console channel rather than to any other device. In addition to being caused by a newline character, a newline is automatically inserted when the cursor reaches the right-hand side of the window; when this happens during an **IOB.SBYT** trap, the error code **ERR.ORNG** (for out of range) is also returned.

If the cursor is suppressed, the newline is held pending. It can be cleared by any call to position the cursor, or activated by any of the following events:

send another byte or string;
changing the character size;
activating the cursor;
requesting the cursor position.

This features allows the right-hand character squares to be used without generating stray blank lines.

The following additional operations apply to screen character output:

IOW.FONT	sets or resets the character fount
IOW.SFLA	sets or resets hardware flash (256-pixel mode only)
IOW.SULA	sets or resets underlining
IOW.SSIZ	sets the character size and spacing

5.3.4 Graphics Operations

The QL can perform line, arc or ellipse drawing on a window basis in scaled coordinates. It also provides a primitive area flood routine. The traps are as follows:

IOG.DOT	draws a point
IOG.LINE	draws a line
IOG.ARC	draws an arc
IOG.ELIP	draws an ellipse
IOG.SCAL	sets the scale
IOG.SGCR	moves the graphics cursor
IOG.FILL	set or reset area filling

5.3.5 Special Properties of Console Channels

For the console device, the **IOB.FLIN** trap behaves in a particular fashion: the characters typed are echoed in the console window, and the left and right cursor keys (with or without CTRL) are used to edit the line in the standard way. In addition, the cursor is automatically enabled.

An additional trap, **IOB.ELIN**, is provided for console channels, which invokes the line editor on a pre-defined string. The line-editor may be exited by typing **ENTER**, or by typing either the cursor-up or the cursor-down character.

The user can temporarily suspend screen output to a console channel by typing the freeze screen character (CTRL-F5). Output is resumed when any character is typed, but the character is ignored for all other purposes. If a finite time-out has been set for the suspended operation, it may return non-complete if the screen is frozen past the time-out period.

5.3.6 Special Keyboard Functions

Several console channels may be open at the same time. If they are used by different jobs, it may be that more than one console channel is expecting input at a given time. When this occurs, the user may cycle round the list of console channels currently expecting input by typing the change queue character on the keyboard. The cursor in the console window to which keyboard input is currently directed will flash if it is enabled. Any enabled cursors in other windows will be steady.

The change queue character is normally CTRL-C (ASCII 3). It can be changed by modifying the system variable **SYS_SWTC**.

The keyboard maintains a type-ahead queue of seven characters in the 8049 processor which controls it. In addition to this, there may be more type-ahead in the queue for each console channel.

The keyboard auto-repeats on all keys except the keyboard change queue character, CTRL-Space (the SuperBASIC BREAK) or CTRL-F5 (the freeze screen character). However, auto-repeat will not occur unless the type-ahead queue for the console channel to which input is currently directed is empty. The delay before auto-repetition begins is held in the system variable **SYS_RDEL**, and the interval between repetitions is held in **SYS_RTIM** (both in multiples of 1/50th or 1/60th of a second). These can be altered by a program.

When CAPSLOCK is pressed, the system will jump to a user-supplied routine whose absolute address is held in the system variable **SYS_CSUB** if the value of this is non-zero. This routine should restore all registers to their initial state before returning.

5.3.7 Extended Operations

A special trap **IOW.XTOP** is provided to allow a program to invoke a user-supplied routine using the same environment that is passed to the routines in the screen driver. See the description in section 15.0 (I/O Traps) for a more detailed discussion of this trap.

6.0 QDOS Device Drivers

A user-supplied Qdos device driver is a collection of routines which allow an applications program to perform IOSS functions on a user-supplied device in the same way as such functions are performed on the devices built into the system. As these routines are linked into the system's lists in front of the corresponding system routines, they may be used to replace the system routines. At the very least, the device driver contains a set of routines for opening a channel, closing a channel, and performing serial I/O on that channel: these routines are called via the IOSS as part of the job that is performing the I/O. The driver may also include one or more tasks, that is, routines performed asynchronously with the calling job, usually under interrupt.

Such tasks, which are known as the physical layer of the device driver, normally communicate with the rest of the device driver, which is known as the access layer, using asynchronous queues. these queues are usually polled by the task at regular intervals, either on every occasion the scheduler is entered, or on every 50/60 Hz polling interrupt.

Drivers for file system devices use a slightly different, and more general, mechanism: this is described in section 7.0.

Both drivers and tasks are linked in to lists provided by the operating system. the following traps are used to add and remove items from those lists:

SMS.LEXI	links in an external interrupt service task
SMS.LPOL	links in a 50/60 Hz polling service task
SMS.LSHD	links in a scheduler loop task
SMS.LIOD	links in a device driver to the I/O system
SMS.LFSD	links in a directory device driver to the file system

SMS.REXI, SMS.RPOL, SMS.RSHD, SMS.RIOD and **SMS.RFSD** remove these links.

The QL provides several utility routines which are useful for various actions commonly performed in device drivers, such as decoding a device name, performing queue operations, etc.

6.1 Device Driver Memory Allocation

Device drivers allocate memory in two areas: the device driver definition block and the channel definition block. The device driver definition block belongs to the driver itself, and is allocated by the code which sets up the driver when it is initialised and linked into the various lists. The channel definition block belongs to each I/O channel, and is allocated by the driver itself when a channel is opened. Various parts of the channel definition block are thereafter used by the IOSS for its own purposes.

In theory, the access layer can allocate space on the heap at other times: in practice this is not usually required. The whole system can be made re-entrant to allow several channels to be open with the same device driver and the same device driver definition block, but with different channel definition blocks.

Note that the system will certainly crash if the area of a channel definition block is deallocated and used for something else before the channel is closed, or if the area of a device driver definition block is deallocated and used for something else before the device driver is removed from the system`s lists, for example if the device driver definition block is in a transient program which is force-removed. This possibility can be obviated by allocating the block in the common heap with a job number of zero, or by allocating it in the resident procedure area.

Tasks must not allocate or release memory: this must be done for them by the access layer, or by the device driver initialisation code.

6.2 Device Driver Initialisation

The code to initialise a device driver must first allocate the space for the device driver definition block, usually by allocating some space in the resident procedure area, although any of the normal allocation mechanisms may be used.

The device driver definition block will normally have the following structure, assuming that A3 has been made to point to it:

$00(A3)	Link to next external interrupt routine
$04(A3)	Address of external interrupt routine
$08(A3)	Link to next poll interrupt routine
$0C(A3)	Address of poll interrupt routine
$10(A3)	Link to next scheduler loop routine
$14(A3)	Address of scheduler loop routine
$18(A3)	Link to access layer of next device driver
$1C(A3)	Address of input/output routine
$20(A3)	Address of channel open routine
$24(A3)	Address of channel close routine
$28(A3)	Any further workspace required for the device driver

The initialisation code should fill in the addresses of the open, close and I/O routines, together with those of any of the routines for tasks that it will be employing. It should also fill in any preset data required in the remainder of the workspace.

Finally, the link routines described above should be called to include the driver in the operating system`s lists.

Note that the structure of the first 24 bytes of the device driver definition block is not mandatory; however it is desirable from the point of view of consistency that it be kept the same. The comments in later sections about the base of the device driver definition block being passed to the driver are only valid if the above structure has been used.

6.3 Physical Layer

The physical layer tasks are normally the ones which perform actual I/O under interrupt or polled control. They usually take data out of queues or put data into queues, the other end of such queues being maintained by the access layer.

When the operating system calls one of the tasks in the physical layer, it passes the task a standard set of values in some of the registers. These values are as follows:

Task service routine

Call parameters		Return parameters	
D1		D1	preserved
D2		D2	preserved
D3	nr. of 50/60Hz Interrupts (sched only)	D3	???
		D4+	all preserved
		A0-A2	preserved
A3	base of device driver def block	A3	preserved
		A4-A5	preserved
A6	system variables	A6	preserved
A7	supervisor stack (64 bytes may be used)		

6.3.1 External Interrupt Tasks

An external interrupt task must check its own hardware to determine whether the interrupt was for itself or for some other driver. It may also need to clear the source of the interrupt at that point. If the interrupt was not for itself, it should return.

6.3.2 Polling Interrupt Tasks

Polling interrupt tasks should only be used when critical timing operations are required. In common with the external interrupt tasks, they can interrupt atomic operations in the rest of the system, such as access layer calls to the same driver, so they should be used with great care.

6.3.3 Scheduler Loop Tasks

Calls from the scheduler loop do not interrupt atomic tasks. This means that operations such as allocating or releasing memory can be performed safely. Note that it is quite common for the same routine to be included both in the scheduler loop and in the external interrupt list.

Scheduler loop tasks are called at around 50/60Hz when the machine is busy, and more frequently if the machine is idle.

All physical layer calls return with RTS.

6.4 The Access Layer

The access layer consists of three routines: the channel open, the channel close, and the Input/Output routine. These routines are called for the appropriate driver by the IOSS in response to a user's trap instruction. In the case of the channel open, the routine is called in turn for each device driver in the machine until a driver's open routine returns correctly to indicate that it has recognised the device name. Due to this mechanism, an incorrect open routine may crash the whole system when an open to any device is attempted, whereas the other routines are only invoked in response to the particular device being used.

All access layer calls return using RTS.

6.4.1 The Channel Open Routine

When the channel open routine is called via the IOSS, the following registers are set:

Channel Open Routine for Device Drivers

Call parameters		Return parameters	
D1		D1	???
D2		D2	???
D3	access key (as per IOA.OPEN)	D3	???
		D4+	???
A0	ptr to device name	A0	channel definition block
		A1-A2	???
A3	base of device driver def block	A3	???
		A4-A5	???
A6	system variables	A6	preserved
A7	supervisor stack (64 bytes may be used)		

Error returns:

Errors as defined below
0 for successful open

The open routine should perform the following operations:

First, decode the name; the utility **IOU.DNAM**, which is described in section 16.0, will normally be used for this purpose. Return with **ERR.ITNF** in D0 if the name was not recognised by this driver, or with **ERR.INAM** if the name was recognised, but some of the additional information was incorrect in value or format.

Then, if the device cannot be shared, check whether the device is in use and prevent another channel from being opened to it. If the device is in use, return **ERR.FDIU**.

Finally, allocate some space for the channel definition block. Any buffers or working area required for each channel are normally allocated in the common heap. Return with **ERR.IMEM** if there was not enough memory to do this.

NOTE: A0 should not be amended by the open routine. D0 must be set to the appropriate error code.

6.4.2 The Channel Close Routine

When this routine is entered, in addition to the usual values of A3, A6 and A7, A0 points to the base of the channel definition block.

Channel Close Routine			
Call parameters		Return parameters	
		D1-D3 ???	
A0	ptr to base of channel definition block	A0	???
		A1-A2 ???	
A3	ptr to base of device driver def block	A3	???
		A4-A5 preserved	
A6	system variables	A6	preserved
A7	supervisor stack (64 bytes may be used)		
Error returns:			
	Always 0, as this routine cannot fail		

The function of the close routine is simply to release the memory taken up by the channel definition block and to ensure that everything in the device driver definition block is tidy.

Under some circumstances, it may not be possible to close the channel immediately because there are bytes waiting to be transmitted by the physical layer. In this case, the physical layer must contain a scheduler loop task, and the close routine should set a flag for the physical layer to complete the release of the memory on the next invocation of that task in which it is possible to do so. When this happens, it is usually necessary to build in a special mechanism to cope with the undesirable event of a program closing a channel to a particular device, and then re-opening it immediately only to receive an "in use" error because the closed channel has not yet been cleared.

NOTE: On completion of the routine D0 must be set to zero as it is assumedthat CLOSE cannot fail. Registers D4 to D7 and A4 to A6 must be set to their initial values before return.

6.4.3 The Input/Output Routine

The I/O routine is called once when an I/O call is made, and then, unless the time-out was set to zero, on every subsequent scheduler loop until the operation is complete or the time-out has expired.

Input/Output Routine

Call parameters		Return parameters	
D0.b	trap code passed to the IOSS		
D1	additional information	D1	updated parameter
D2	additional information	D2	???
D3	0 for first call, else -1	D3	???
		D4+	???
A0	ptr to base of channel definition block	A0	preserved
A1	additional information	A1	updated parameter
A2	additional information	A2	???
A3	ptr to base of device driver def block	A3	preserved
		A4-A5	preserved
A6	system variables	A6	preserved
A7	supervisor stack (64 bytes may be used)		

Error returns:

All returns defined by the IO traps

The I/O routine should return **ERR.NC** (not complete) if it cannot complete the operation immediately. If a string operation has been partially completed, the values in D1 and A1 (number of bytes transferred and buffer pointer) should be set appropriately so that the operation can continue on the next try. D0 should be zero on return if the operation has been completed correctly.

Since most of the code for handling serial I/O is common to all device drivers, the I/O routine usually calls one of the utility routines **IOU.SSQ** or **IOU.SSIO** (which are described in section 16.0). **IOU.SSQ** assumes that the only function of the access layer is to move bytes in and out of a pair of queues pointed to by fixed positions in the channel definition block, while **IOU.SSIO** assumes that the operations required of it can all be made up out of three primitive routines for sending one byte, fetching one byte, and checking for pending input, such routines being supplied by the writer of the device driver.

Note that channels are assumed to be bidirectional; it is the responsibility of the I/O routine to trap an operation in a direction that is not allowed. Note also that output operations which appear to the user as complete have merely completed the access layer call correctly: there being no general way in which the user can ascertain whether the physical layer has in fact completed the operation.

NOTE: On completion of the routine, registers A0, A2 to A6 (inclusive) should be reset to their initial values before return.

7.0 Directory Device Drivers

Drivers for devices which have a directory and form part of the filing system have a somewhat extended set of functions. For directory device drivers, there are three blocks in which memory is allocated, rather than two: these are the directory driver linkage block, the physical definition block and the channel definition block.

There is one directory driver linkage block for each directory driver: it is an extended form of the device driver definition block as found in a non-directory device driver. The block contains information about how to use the driver, together with the links in the operating system's lists.

Each directory driver may control up to 8 drives (numbered 1 to 8). Each drive has one physical definition block: this contains the drive number and information about the medium.

For each I/O channel that is open, there is an open channel definition block.

The file system is assumed to be composed of 512-byte blocks: thus a byte within a file is addressed by the IOSS by a block number and a byte number within that block. It is of course possible to have a different physical block size, but the mapping of the IOSS structure onto the physical structure will be less convenient.

Each file is assumed to have a 64-byte header (the logical beginning of file is set to byte 64, not byte zero). This header should be formatted as follows:

$00	long	file length
$04	byte	file access key (used by third parties software)
$05	byte	file type
$06	8 bytes	file type-dependent information
$0E	2+36 bytes	file name
$34	long	update date [EXT,DD2]
$38	word	version number [DD2]
$3A	word	reserved
$3C	long	backup date [DD2]

The current file types allowed are: 2, which is a relocatable object file; 1, which is an executable program; and 0 which is anything else. In the case of file type 1, the first longword of type-dependent information holds the default size of the data space for the program.
For level 2 devices, a type of -1 (or 255 decimal) stands for a subdirectory.

7.1 Initialisation of a Directory Driver

The initialisation routine should first allocate room for the directory driver linkage block, and then write into it the information about the driver routine addresses, the length of the physical definition block required for each drive, and the drive name. Note that for directory drivers, the decoding of the device name is performed by the IOSS, not by the open routine in the device driver as in non-directory drivers: the function of the open routine is to search for the file name within the given drive. The linkage block may be allocated in the resident procedure area if the driver is resident there, but will usually be in the common heap. The system will crash if the linkage block is overwritten without the driver being unlinked.

When this has been done, the traps **SMS.LEXI, SMS.LPOL, SMS.LSHD** and **SMS.LFSD** can be called to link the driver and any associated tasks into Qdos.

The format of the directory driver linkage block is as follows (assuming that A3 has been made to point to it):

iod_xilk	$00(A3)	link to next external interrupt routine
iod_xiad	$04(A3)	address of external interrupt routine
iod_pllk	$08(A3)	link to next 50/60 Hz interrupt routine
iod_plad	$0C(A3)	address of 50/60 Hz interrupt routine
iod_shlk	$10(A3)	link to next scheduler loop routine
iod_shad	$14(A3)	address of scheduler loop routine
iod._olk	$18(A3)	link to access layer of next directory driver
iod_ioad	$1C(A3)	address of input/output routine
iod_open	$20(A3)	address of channel open routine
iod_clos	$24(A3)	address of channel close routine
iod_iend		end of minimum device driver linkage
iod_fslv	$28(A3)	address of entry for forced slaving
iod_spr1	$2C(A3)	reserved
iod_cnam	$30(A3)	address of set channel name [SMSQ]
iod_frmt	$34(A3)	address of entry to format medium
iod_plen	$38(A3)	length of physical definition block
iod_dnus	$3C(A3)	word-length of drive name, characters of drive name (e.g.MDV) current usage
iod_dnam	$42(A3)	word-length of drive name, characters of drive name real name [SMSQ]

Note that a directory driver must have at least 40 bytes of RAM for the linkage block.

For additional SMSQ features please refer to section 18.9

7.2 Access Layer

The access layer of a directory driver contains five routines: the channel open/file delete routine, the close routine, the I/O routine, the forced slaving routine and the format routine.

For all directory device driver access layer calls (including open), A0 points to the base of the channel definition block when each routine is called. However, the format of the block is somewhat different.

The first $18 bytes are reserved for the IOSS (heap entry header). The format of the block for microdrives is:

$18(A0)	**CHN_LINK**	long	link to next file system channel
$1C(A0)	**CHN_ACCS**	byte	access mode (D3 on open call, -ve on delete)
$1D(A0)	**CHN_DRID**	byte	drive ID
$1E(A0)	**CHN_QDID**	word	number of file on drive
$20(A0)	**CHN_FPOS**	word	block number containing next byte
$22(A0)		word	next byte from block
$24(A0)	**CHN_EOF**	word	block number containing byte after EOF
$26(A0)		word	byte after EOF
$28(A0)	**CHN_CSB**	long	pointer to slave block table for current slave block which may hold current/ next byte
$2C(A0)	**CHN_UPDT**	byte	file updated
$32(A0)	**CHN_NAME**	2+36 bytes	file name
$58(A0)		72 bytes	spare

Section 18 contains details of the block for other filing systems.

A1 points to the physical definition block, which is formatted as follows:

The first $10 bytes are reserved for the IOSS (heap entry header).

$10(A1)	**FS_DRIVR**	long	pointer to access layer link for driver
$14(A1)	**FS_DRIVN**	byte	drive number
$16(A1)	**FS_MNAME**	2+10 bytes	medium name
$22(A1)	**FS_FILES**	byte	number of files open on this medium

The physical format for the microdrive system can be found in section 18.

7.2.1 The Channel Open/File Delete Routine

The function of the open routine depends on the access mode. This may have been passed to the IOSS in D3 if the open routine was called as a result of an **IOA.OPEN** trap, or it may be a negative number, which would be the case if the routine has been entered as a result of an **IOA.DELF** trap.

In order to understand the open routine, it is necessary first to understand the way in which Qdos handles device names. When a device name is passed to the IOSS as a result of an open or delete call, the IOSS looks for a match in its lists of device drivers and directory device drivers. The matching mechanism for non-directory device drivers is defined within the open routine for that driver. The matching mechanism for directory device drivers is as follows. The first characters of the name are checked against the driver name in the directory driver linkage block (which is put there when the driver is initialised), and these are expected to be followed by a drive number between 1 and 8, followed by an underscore, followed usually by the filename. If a match is found, the file system looks to see if there is a physical definition block for that drive already in existence. If there is not, a physical definition block is created in the system's table of physical definition blocks (the drive ID in the channel definition block is an index to this table). Note that the file system has no knowledge of whether a drive is actually connected, and will set up the definition block regardless.

The IOSS then checks to see if this is the second or subsequent open to a shared file: if this is the case it generates the complete channel definition block itself, setting **CHN_FPOS+2** to $40 (i.e. the first byte behind header), and copies the remaining information from the channel definition block for the first open. The directory driver's open routine is not called. Otherwise, the IOSS calls the open routine, passing it the file name in the channel definition block.

Channel Open Routine for Directory Device Drivers

Call parameters		Return parameters	
D1		D1	???
D2		D2	???
D3		D3	???
		D4+	all preserved
A0	base of channel definition block	A0	preserved
A1	base of physical definition block	A1	preserved
A2		A2	???
A3	base of device driver def block	A3	preserved
		A4-A5	???
A6	system variables	A6	preserved

Error returns:

Errors as defined below
0 for successful open

The channel and physical definition blocks are all set to zero except for the following, which are filled by the IOSS:

CHN_LINK	link to next file system channel
CHN_ACCS	access mode
CHN_DRID	drive ID
CHN_NAME	file name
FS_DRIVR	pointer to directory driver access layer
FS_FILES	number of files open on this drive (maintained by IOSS)

In the case of a device with removable media, the open routine should find out the name of the medium and install it in **FS_MNAME**. It should also look at the access mode to find out which operation is required. If the required operation is delete, it should perform that operation and return, but if the required operation is another sort of open, then it should fill in the appropriate portions of the channel definition block, namely **CHN_QDID, CHN_EOF, CHN_EOF+2, CHN_FPOS** and **CHN_FPOS+2. CHN_CSB** is a pointer to the slave block table which may be filled in as an indication to the I/O routine that the block it is looking for may be slaved there. The I/O routine must check this however, normally by searching the slave table.

The IOSS will free the channel definition block on exit from the open routine if the action was a delete or if the open routine returns an error key in DO.

The maintenance of the directory structure of the medium is the responsibility of the open and close routines- the IOSS plays no part in this. Equally, the open routine is responsible for understanding the meaning of the access mode and reacting accordingly.

NOTE: A6 should be reset to its initial state before return.

7.2.2 The Channel Close Routine

As far as the IOSS is concerned, this routine behaves in the same way as for a non-directory device driver. It is of course necessary for the close routine to maintain the directory structure of the medium, so its operation will normally be rather more complicated.

The close routine for a directory device driver has two additional functions: it must unlink the channel from the list of files open, and must decrement the **FS_FILES** field in the physical definition block, which gives the number of files open on the medium. Suitable code for performing these operations and ending the close routine is as follows:

```
* get address of physical definition block into A2
    MOVEQ     #0,D0               top three bytes must be clear
    MOVE. B   CHN_DRID(A0),D0     get the drive ID
    LSL.B     #2,D0               convert it to a table offset
    LEA.L     SYS_FSDD(A6),A2     get base of PDB table
    MOVE.L    (A2,D0.W),A2        get address from (base+offset)
* now decrement the file count
    SUBQ.B    #1,FS_FILES(A2)
* now unlink the file
    LEA       CHN_LINK(A0),A0     get address of link pointer . . .
    LEA       SYS_FSDT(A6),A1     . . . and pointer to start of linked list
    MOVE.W    MEM.RLST,A4         routine to unlink an item
    JSR       (A4)
    LEA       -CHN_LINK(A0),A0    restore A0 to base of channel def
    MOVE.W    MEM.RCHP,A4         routine to release channel def space
    JMP       (A4)                call it, and exit from the close
```

The close routine must also initiate the process of tidying up any slave blocks remaining for that channel. It need not force the slave blocks to be made into true copies itself, but it must be guaranteed that the copying will happen without further intervention by the calling program.

7.2.3. The Input/ Output Routine

This routine also appears to the IOSS to be identical for both directory and non-directory device drivers, though once again the routine is usually rather more complex for most normal file system devices. The main difference is that the I/O routine for a random access file system device must take into account the current block and position as provided by the IOSS, since these may have been updated by the IOSS as a result of a file pointer positioning trap.

7.3 Slaving

The area of memory between **SYS_FSBB** and **SYS_SBAB** is used by the filing system as temporary storage for file slave blocks and for the slave block table. A slave block is a block of 512 bytes of data. The slave block table is a table of entries sized 8 bytes whose start point is held in the system variable **SYS_SBTB** and whose top is held in the system variable **SYS_SBTT**; the system variable **SYS_SBRP** points to the most recently allocated slave block table entry. The address of a slave block, relative to the base of system variables, is equal to 512/8 times the offset of the corresponding entry in the slave block table from the beginning of that table.

Currently, only the first byte of each slave block table entry is used by Qdos itself: the remaining bytes are available for use by the driver. This byte is divided into two four-bit nibbles. The most significant nibble contains the drive identifier (0..15), and the least significant nibble is a code indicating the status of the block. The byte is formatted as follows:

$00	unavailable to filing system
$01	empty block
$x3	block is true representation of file
$x7	block is updated, awaiting write
$x9	block is awaiting read
$xB	block is awaiting verify

x is the drive ID for this file

For Microdrives, the remaining space in each slave block table entry is laid out as follows:

SBT_PRIO	01	byte	available for slaving algorithms
SBT_SECT	02	word	physical sector number *2
SBT_FILE	04	word	file number
SBT_BLOK	06	word	block number within the file

Section 18.6 contains details of table entries for other devices.

It is left the device driver to decide what the slave blocks are used for but it must be prepared to release a slave block if requested to do so by the memory manager. This is done by calling the driver's forced slaving routine with the following parameters:

Forced Slaving Routine

Call parameters		Return parameters	
D1		D1	???
D2		D2	???
D3		D3	???
		D4+	all preserved
A0		A0	???
A1	base of offending slave block	A1	???
A2	physical definition block	A2	???
A3	base of device driver def block	A3	preserved
		A4+	preserved

This routine cannot fail.

Typically the slave blocks are used to buffer data being written to a device, the actual writing being carried out by an asynchronous task.

Searching for an empty slave block involves performing a linear search through the slave block table, usually starting from **SYS_SBRP** or **SYS_SBTB**. The status of each entry in the table must be checked and only those blocks which are empty or true representations should be taken. When a new block is allocated **SYS_SBRP** should be updated to point to the allocated block. Allocating slave blocks is a form of memory allocation and should only be carried out by access layer or scheduler loop calls.

This position in memory of a slave block which corresponds to a slave block table entry may be calculated using the following code:

```
     MOVE.L    A4,D0              A4 is pointer to slave block table entry
*
* form offset into slave block table, gives slave block no.*8
* entries are 8 bytes wide in table
*
     SUB.L     SYS_SBTB(A6), D0
     LSL.L     #6,D0              multiply by 64 (8*64=512)
     MOVE.L    D0,A5
     ADD.L     A6,A5              add offset to system variable base
* A5 now has base address of slave block
```

7.3.1 The Format Routine

This routine is to a large extend independent of the other routines. It is called with the drive number in D1, a pointer to the medium name in A1, and a pointer to the directory driver linkage block in A3.

Format routine

Call parameters		Return parameters	
D1	drive number	D1	number of good sectors
D2		D2	total number of sectors
		D3+	???
A0		A0	???
A1	ptr to medium name	A1	???
A2		A2	???
A3	base of device driver def block	A3	???
		A4-A5	???
A6	system variables	A6	preserved

Error returns:

FMTF format failed

8.0 Built-in Device Drivers

The following devices are built in to the QL ROM:

CON_wXhAxXy_k — Console I/O,
window area "w" by "h" pixels, top left hand corner at pixel position "x", "y",
keyboard type-ahead buffer length "k" characters.
The size and position are defined in terms of pixels on a 512x256 display map (position 256x128) is the centre of the screen in both display modes).
Default `CON_448x200a32x16_128`

SCR_wXhAxXy — Screen output
window definition is as for CON.
Default `SCR_448x200a32x16`

SERnphz — RS232 serial I/O
port "n",
"p" indicates parity: E, O, M, S for even, odd, mark, or space parity,
"h" indicates handshaking, H to enable it, I if it is to be ignored
"z" indicates protocol:
  R indicates raw data,
  Z or C indicates that CTRL-Z is used as an EOF marker,
  C indicates that ASCII 13 is to be exchanged with ASCII 10 on input and vice versa on output.
Default `SER1HR` no parity.

NETI_nn — Serial network input
link from node "nn"

NETO_nn — Serial network output
link to node "nn"

PIPE_n — Job connection and synchronisation
if "n" given it is an output pipe of length n bytes,
otherwise it is an input pipe connected to the channel ID passed in D3.

MDVn_name — Microdrive file
MDV1 refers to Microdrive "1".

FLPn_name — Floppy Disc file [EXT]
FLP1 refers to Floppy Disk "1".

Within device names, no distinction is made between upper and lower case letters.

Floppy Disks are supported in a standard way. The format and additional facilities of the floppy disk driver are explained in section 8.1 and 8.2. For the extended drivers of the QL Emulator, their additional parameters and facilities, refer to the Emulator's manual.

8.1 QL Floppy Disc Format [EXT]

For ease of data transfer between different manufacturer's floppy disc systems, it is necessary to have a common standard of disk formats. Clearly this only applies where the discs are physically compatible: physical dimensions, recording method, recording density, track spacing and positioning must all match on the source and destination machines. There is no requirement for the format for (e.g.) 5.25" and 8" discs to be the same, however, for convenience, this standard is proposed not only for 5.25" drives, but also for electrically compatible 3.5" and 3" drives. Similar formats may be derived for other standards. This standard has been based on the original Sinclair Research proposals, and compatibility between different manufacturers has already been established.

Floppy disks will be sectored in 512 byte sectors. 5.25" compatible disks will have 9 sectors per track (MFM 200ms rotation), for a 40 track drive, single sided, this gives 180k bytes and for an 80 track drive, double sided, this gives 720k bytes capacity.

Tracks are numbered from 0, sectors on a track are numbered, by ones, from sector 1 immediately after the index mark.

The physical format is basically IBM System 34 (8" MFM) with four changes. There is no index mark recorded, the sector length flag is $02, the data record is 512 bytes long, and the write splice gap is increased.

For IBM standard format on MFM recording with 256 bytes sectors, the write splice gap at the end of a data record is 54 bytes. This is increased to 84 bytes allowing for a short term speed variation of + or - 4%. Using this, each sector is recorded in 658 bytes, this sets the gap between sector 9 and 1 to approximately 6250-5922 (328) bytes, allowing a long term speed variation of + or - 2.75%.

Regardless of the physical characteristics, all floppy disks will have the same directory structure. Track zero will hold the map of sector allocations. The first block of the map will be in sector 1 side 0 track 0.

The first 96 bytes of the sector map hold information about the format of the rest of the drive:

q5a_id	$00	long	format ID
q5a.id	'QL5A'		
q5ax.id	'QL5B'		as QL5A but no physical-logical translation
q5a_mnam	$04	10*bytes	medium name (space filled)
q5a_rand	$0e	word	random number set during format
q5a_mupd	$10	long	count of updates
q5a_free	$14	word	free sectors
q5a_good	$16	word	good sectors
q5a_totl	$18	word	total sectors (sectors*tracks)
q5a_strk	$1a	word	sectors per track (<=9)
q5a_scyl	$1c	word	sectors per cylinder (e.g. 9 or 18)
q5a_trak	$1e	word	number of tracks (cylinders)
q5a_allc	$20	word	allocation size (sectors per alloc group)
q5a_eodr	$22	long	current end of directory (blocck/byte format)
q5a_soff	$26	word	sector offset
q5a_lgph	$28	18 bytes	logical to physical sector translate
q5a_phlg	$3a	18 bytes	physical to logical sector translate (standard)
q5a_spr0	$4c	20 bytes	$ff
q5a_gmap	$60		3 byte entry map in form: (file id-1) / Group number
q5a_mtop	$600		

The map is always of a size to fill the first three (logical) sectors of the drive, being padded with 'non-existant' sectors if necessary to fill the (512*3-96)/3=480 allocation allowed. This is adequate for up to 720k bytes with a sector allocation size of 3. (3 groups per track per side), and a sector allocation size of 6 for up to 1440k bytes. For extended density disks, the number of entries in the map is 1600, therefore the size is 1600*3+96=6144.

The format ID is a 4 byte ID indicating that the format conforms to this standard.

The medium name, random number and update count are used to provide protection against media change. In addition the update count allows detection of the case of a medium being removed, updated on another machine or drive, and being re-inserted into the original drive.

The drive statistics are maintained in the map header for simplicity and speed of access, while the directory EOF is maintained in the map to reduce the access overheads associated with directory handling.

Sectors are allocated to files in multiples of the allocation size. To ensure fast serial access, it is necessary to space adjacent blocks of a file in such a way as to allow processing between those blocks. The translate tables define the spacing. There is an additional overhead on accessing a sector on a new track, and so there is an additional offset to be applied to the sector calculation for each track.

The logical sector is obtained from the sector map by the following calculation:

(sector in map * alloc size + sector in alloc group) MOD sectors per cylinder

In the logical to physical translate table, the MSB of the translate byte indicates the side number, while the remaining 7 bits give the sector number (numbered from 0 to 8). In the physical to logical translate table the first nine bytes correspond to sectors 0 to 8 on side 0, and the next 9 bytes to sectors 0 to 8 on side 1. (Note that the internal numbering of sectors on a track starts at 0 for convenience in calculation: 1 is added to the sector number immediately before recording or reading).

E.g. for a 1 in 3 interleave, 18 sectors per cylinder, the tables will be:

```
00 03 06 80   83 86 01 04   07 81 84 87   02 05 08 82   85 88
00 06 0c 01   07 0d 02 08   0e 03 09 0f   04 0a 10 05   0b 11
```

For each track there will be an additional offset to allow for steps between adjacent tracks. So the final physical sector is calculated as

(translated sector + track * sector offset) MOD sectors per track

The EOF of a file is the position of the next byte after the end of the file. Thus for an empty file it is 0/40. The block number starts at 0, the byte number is between 0 and $1ff inclusive.

The allocation map itself is a table giving the usage of each group of sectors. For each group there are three bytes: the file number in the first 12 bits and in the second twelve bits, the numbers of the blocks of the file, stored in the group, divided by the allocation size. Thus for file number 2, the first allocation of sectors is identified in the map as 002000, the next allocation as 002001 and so on.

The file number is the index into the master directory. The file numbers are allocated as follows:

000	Master directory
001+	Normal files
F8x	Sector map
FDx	Vacant sector group
FEx	Bad sector group
FFx	Non existant sector group

The master directory is a table of file headers in standard format. The first 64 bytes of any file do not contain any useful information.

8.2 Direct Sector Read/Write [EXT]

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

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

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

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

When this file is open, no other file may be open on the drive. The only IO calls supported for this type of file are **IOB.FMUL, IOB.SMUL, IOF.POSA** and **IOF.POSR** (D0=$03, $07, $42 or $43), to read or write complete sectors or to set the position. The parameter (D1) to the **POSR** call is ignored, but the current position is returned. Reading or writing a sector does not change the file position.

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

sector number + side * 256 + track * 65536

To ensure compatibility with string IO the length specified in the **SMUL** and **FMUL** calls may be one of three values:

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

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

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

Direct sector read/write calls can be used for a 40 track disk in an 80 track drive by multiplying the track counter by two.

8.3 Additional Standard Device Drivers [ST] [EXT]

In addition to the standard device drivers exist some other devices and directory devices which are defined for a whole range of machines, including SMS2. Application software should allow these optional devices whenever possible. As most device do not need special treatment, this should be no problem at all.

FLPn_name Floppy Disc file
FLP1 refers to Floppy Disk "1".

RAMn_name RAM Disc file
RAM1 refers to RAM Disk "1".

WINn_name Harddisk or Changeable Disk file
WIN1 refers to Harddisk "1".

The Serial and Parallel Port drivers accept additional parameters:

SERnpftce Serial Port receive and transmit
SRXnpftce Serial Port receive only
STXnpftce Serial Port transmit only
PARntce Parallel Port (transmit only)
n - port number e.g. 1 or 2; default is 1
p - parity: O (7 bit + odd parity), E (7 bit + even parity),
M (7 bit + mark=1), S (7 bit + space=0); default is none
f - flow control: H (Hardware CTS/DTR), I (Ignore flow control),
X (XON/XOFF); default H
t - translate: D (direct output), T (translate), A (auto-CR)
c - <CR>: C (<CR> is end of line), R (no effect)
e - end of file: F (<FF> at end of file), Z (CTRL Z at end of file)

PRT Printer Port (either SER or PAR)

NULF Null device, emulating null file.
NULZ emulates a file filled with zeros.
NULL emulates a file filled with null lines.
NULP always returns "not complete".

Named pipes have been added to the unnamed type:

PIPE_name_n Job communication and synchronisation
if "n" given it is an output pipe.

9.0 Interfacing to SuperBASIC

When writing SuperBASIC procedures or functions in machine code, there are several things that an applications programmer may want to do: he may wish to look at or modify the information held in SuperBASIC variables and arrays, he may wish to access or modify the SuperBASIC list of I/O channels, and he may wish to reserve and use space on the arithmetic stack. He will also, of course, wish to access the list of parameters passed to the routine and return values either to those parameters or in a function return. In order to do this, it is necessary to understand the data structures used by the interpreter and to emulate the interpreter's techniques for manipulating them.

9.1 Memory Organisation within the SuperBASIC Area

The SuperBASIC area contains twelve distinct areas:

the job header,
the SuperBASIC work areas,
the name table,
the name list,
the variable values area,
the channel table,
the arithmetic stack,
the token list,
the line number table,
the program file,
the return list,
the buffer.

There are also various other stacks used by the interpreter.

The job header is located at the bottom of the SuperBASIC area, and looks just like other job header (see section 18.5). Immediately above this is the SuperBASIC work area; this is an area of fixed storage used for the working variables of the interpreter. Included in these working variables are pointers to the other areas: the interpreter can not only shuffle these areas around, but may also ask Qdos to change the size of the whole SuperBASIC area.

The organisation of this area is shown in section 18.3. Throughout normal operation of the interpreter, A6 points to the base of the SuperBASIC work area, the whole of which may move between instructions, with a corresponding change in A6. All the pointers are, of course, relative to A6, so that their values need not be changed when the SuperBasic area is moved.

The name table, the name list and the variable values area are required by the applications programmer in order to access and/ or modify SuperBASIC variables and parameters. The channel table is required in order to access SuperBASIC I/O channels, and the arithmetic stack (usually abbreviated to RI stack) is a convenient area in which to reserve storage, and is also where parameters are passed. The remaining areas are not described in this document.

9.2 The Name Table

All variables, procedure names, parameters and even expressions are handled through the name table. This is a regular table of eight byte entries, but the entries hold different information according to the type of entry.
The entries may be as follows:

Bytes 7-4	Bytes 3-2	Bytes 1-0	Type
Value pointer	Name pointer	$0001	Unset string
Value pointer	Name pointer	$0002	Unset floating point number
Value pointer	Name pointer	$0003	Unset integer
Ptr to RI stack	-1	$0101	String expression
Ptr to RI stack	-1	$0102	Floating point expression
Ptr to RI stack	-1	$0103	Integer expression
Value pointer	Name pointer	$0201	String
Value pointer	Name pointer	$0202	Floating point number
Value pointer	Name pointer	$0203	Integer
Value pointer	-1	$0300	Substring
Value pointer	Name pointer	$0301	String array
Value pointer	Name pointer	$0302	Floating point array
Value pointer	Name pointer	$0303	Integer array
Line no in msw	Name pointer	$0400	SuperBASIC procedure
Line no in msw	Name pointer	$0501	SuperBASIC string function
Line no in msw	Name pointer	$0502	SuperBASIC f.p. function
Line no in msw	Name pointer	$0503	SuperBASIC integer function
Value pointer	Name pointer	$0602	REPeat loop index
Value pointer	Name pointer	$0702	FOR loop index
Abs. address	Name pointer	$0800	Machine code procedure
Abs. address	Name pointer	$0900	Machine code function

Byte 0 of the name table has an additional usage during parameter passing: see section 9.8.

The Name pointer is a pointer to an entry in the name list (see the following section). A name pointer of -1 indicates a nameless item such as the value of an expression; any other negative pointer indicates a pointer to another entry in the name table of which this entry is a copy.

The Value pointer is a pointer to an entry in the variable values area (see section 9.4). A value pointer of -1 indicates that the value is undefined.

Since all these areas may move during execution, the pointers are offsets from the base of each area. For the RI stack, the base is at the high address; for the others it is at the bottom.

Note that functions written in SuperBASIC are typed according to whether the name ends in %, $ or neither. Functions written in machine code, in common with procedures written in SuperBASIC or machine code, have no type.

The entries for expressions and substrings are for use within the expression evaluator: the applications programmer would not normally use them.

9.3 Name List

The names in the name list are stored as a byte character count followed by the characters of the name. Note that this format is different from all the other uses of strings in Qdos in which a word character count is used.

9.4 Variable Values Area

This area is a heap in which the values are stored. The format for each type of data item is given in the following sections.

9.5 Storage Formats

9.5.1 Integer Storage

An integer is a 16-bit two's complement word.

9.5.2 Floating Point Storage

A floating point number is stored as a two-byte exponent followed by a four-byte mantissa.

The most significant four bits of the exponent are zero, whilst the remaining twelve bits are an offset from -$800. The mantissa is two's complement and fractional, with bit 31 of the mantissa representing -1, and bit 30 of the mantissa representing +1/2. There are no implicit bits in the mantissa, so either bit 31 or bit 30 will be set for a normalized number, except in the special case of zero.

The value of the number is thus **mantissa*2 to the power (exponent-$800)**. If the mantissa is viewed as two's complement absolute (as opposed to fractional), the value of the number is given by: **mantissa*2 to the power (exponent-$81F).** The $1F corresponds to 31 decimal: the length of the mantissa minus one.

Examples of floating point storage are as follows:

Hex	Decimal value
0804 50000000	10.00
0801 40000000	1.00
07FF 40000000	0.25
07FF 80000000	-0.50
0800 80000000	-1.00
0000 00000000	0

9.5.3 String Storage

A string is stored as a word character count, followed by the characters of the string. The string storage always takes a multiple of two bytes. Examples are as follows:

Hex	String
0004 41424344	"ABCD"
0003 414243xx	"ABC"
0000	""

9.5.4 Array Storage

An array descriptor has a header which consists of a longword offset of the array values from the base of the variable value area, followed by the number of dimensions (word), followed by a pair of words for each dimension. The first word is the maximum index, the second word is the index multiplier for this dimension.

The storage of floating point and integer arrays is entirely regular. A floating point array takes 6 bytes per element, an integer array 2 bytes per element.

A string array is stored as an array of characters; except that the zeroth element of the final dimension is a word containing the string length. The final dimension defines the maximum length of the string. This is always rounded up to the nearest even number.

A substring is the result of internal slicing operations; this is a regular array of characters; the base of the indexing is one rather than zero.

```
Examples of Floating Point Storage
Floating point variables (in hex)
0000 0000 0000                0.0
0801 4000 0000                1.0
0800 8000 0000                -1.0
0804 5000 0000                10.0

Floating point arrays
      base,2,3,3,2,1          DIM A(3,2)

Examples of string storage (numbers in decimal)
String variable
      4;65,66,67,68           "ABCD"
String array
      base,2,3,12,10,1        DIM A$(3,10)
      4;65,66,67,78,x,x,x,x,x,x   "ABCD"
      9;49,50,51,52,53,54,55,56,57,x   "123456789"
      0;x,x,x,x,x,x,x,x,x,x   ""
      1;32,x,x,x,x,x,x,x,x,x  " "
Substring array
      base,1,3,1              A$(0,1 TO 3) as above
      65,66,67                "ABC"
```

9.6 Code Restrictions

There is a simplest set of rules for writing procedures in machine code for SuperBASIC:

1. As the SuperBASIC program area is liable to move at any time while the execution is in user mode, all references to this area must be indexed by A6 or A7. A6 and A7 must never be saved, used in arithmetic or address calculations, and must never be altered, except by pushing or popping the A7 stack. In extreme circumstances it is possible to enter supervisor mode (TRAP #0) to make the following action atomic. If this is done, A6 and the user stack pointer must not be saved or manipulated before entering supervisor mode, and they must be restored before exiting.
2. Not more than 128 bytes must be used on the user stack.
3. D0 must be returned as an error code (long).
4. D1 to D7 and A0 to A5 inclusive may be treated as volatile.

9.7 Linking in New Procedures and Functions

New SuperBASIC procedures and functions written in machine code may be linked into the name table using the vectored routine **SB.INIPR** (see section 16.0). When the procedures and functions are in a ROM in the suitable format (see section 11.4), **SB.INIPR** is called automatically. If the procedures and functions are to be stored in RAM, they should be loaded into the resident procedure area as, once added, they may not be removed except by re-booting the machine. It is usually convenient to load the code for calling **SB.INIPR** to make the linkage into the same area, although this is not necessary.

9.8 Parameter Passing

The SuperBASIC interpreter passes parameters using a substitution mechanism, which operates as follows. The interpreter first evaluates any of the parameters that are expressions. A new entry is then created at the top of the name table for each actual parameter. In the case of a procedure or function written in SuperBASIC, this is followed by a null entry for any formal parameter that is missing from the actual parameter list. The interpreter then swaps the new name table entries with the old name table entries corresponding to the actual parameters. In the case of a procedure or function written in machine code, the code is then called with A3 pointing to the name table entry for the first parameter in the list, and A5 pointing to the last ((A5-A3)/8 is the number of parameters).

If a local statement is encountered, the entry in the name table is copied to a new position at the top of the table, and an empty entry put in its place.

At the end of a SuperBASIC procedure or function, the parameter entries are copied back and local variables are removed. The parameter entries in the name table together with any temporary storage in the variable value table are then removed for all procedures and functions.

Byte 0 of the name table entry for a parameter has an additional meaning to that associated with a normal name table entry. The bottom four bits have the usual indication of type (0=null, 1=string etc.), but the top four bits are used to indicate the separator that was present after the parameter in the actual parameter list, together with information as to whether the actual parameter was preceded by a hash (#).

Thus the format of byte 0 is as follows:

h sss tttt

tttt: type: 0=null, 1=string, 2=floating point, 3=integer

sss: type of following separator: 0=none, 1=comma, 2=semi-colon, 3=backslash, 4=exclamation mark, 5=TO

h: 1 if the parameter was preceded by hash, otherwise 0

Note that byte 0 of the name table is located at 1(a3) as it is part of a word (see section 9.2).
The name pointer of a parameter (if it is not an expression or substring) is the index of the name table entry of the item from which it is copied. Thus the parameter "name" can be obtained from the name list entry of that item (see also section 9.9). The index must be multiplied by the entry size (8) to get the pointer.

9.9 Getting the Values of Actual Parameters

For the purpose of using scalar (as opposed to array) parameters locally in the same way as "call by value" parameters in other high-level languages, it is expedient to use one of a set of four vectored routines which place the values of actual parameters on the arithmetic stack. Each routine assumes that all the parameters will be of the same type. It is passed the values of A3 and A5 which point to the name table entries for the parameters; it returns the number parameters fetched in the least significant word of D3, and the values themselves in order on the arithmetic stack with the first parameter at the top (lowest address) of the stack. These routines smash the separator flags. They are as follows: **SB.GTINT** gets 16-bit integers, **SB.GTFP** gets floating point numbers, **SB.GTSTR** gets strings, and **SB.GTLIN** gets floating point numbers but converts them to 32-bit long integers.

These routines may still be used when processing parameters of mixed type or when wishing to inspect the separators. To begin with, the values of A3 and A5 should be saved; then, for each parameter in the succession, the separator flags are inspected, and the appropriate routine is called with A3 pointing to the parameter and A5 equal to A3+8, thus getting one parameter.

These routines smash D1, D2, D4, D6, A0 and A2. The error codes are returned in D0 and the condition codes.

A special technique is provided for use in those routines in which it is necessary for the user to be able to type in a string without quotes, as it's required for SuperBASIC commands involving device names. Firstly, the name is inspected to see if it is a valid set string variable. If it is, the string is fetched using **SB.GTSTR**; if it is not, the parameter's name itself is fetched from the name list, and converted to string form by changing its word count from byte to word, realigning the string if necessary. If a string is to be input without quotes, it must of course follow the rules for SuperBASIC names, as described in the **Concepts** manual.

9.10 The Arithmetic Stack Returned Values

The top of the arithmetic stack is usually pointed to by A1. Space may be allocated on the stack by calling the vectored routine **QA.RESRI**: the number of bytes required is given in D1.L; D0 to D3 are smashed by the call. Since both the stack within the SuperBASIC area and the SuperBASIC area itself may move during a call, the stack pointer should be saved in **BV_RIP(A6)** before the call, and restored from **BV_RIP(A6)** after the call has been completed. The routine ensures that the restored value will be correct.

The vectored routines for getting parameters reserve their own space on the arithmetic stack.

The arithmetic stack is automatically tidied up both after procedures, and after errors in functions. To make a good return from a function, the returned value should be at the top (lowest address) of the stack with nothing below it (that is with both (A6,A1.L) and **BV_RIP(A6)** pointing to it) when the routine is exited. The type of the returned value should be in D4 (1=string, 2=floating point, 3=integer). Since SuperBASIC has no long integer type, long integers must be converted to floating point before returning.

Values can also be returned to parameters or, indeed, global variables, by putting the value on the arithmetic stack in the same way, pointing A3 to the appropriate name table entry and calling the vectored routine **SB.PUTP**. D0 is an error return, and D1, D2, D3, A0, A1 and A2 are smashed. If the actual parameter was an expression, no error will be given, but the value returned will be lost. The type of the returned parameter is determined by the name table entry, and the information on the arithmetic stack must be in the correct form.

As functions do not tidy up the arithmetic stack automatically unless an error occured, it is very important to make sure that the stack does not grow on function returns, especially if strings have been passed and returned. Also, the routine **QA.RESRI** does not update A1 (return value undefined!) or move the stack, it just makes sure that enough memory is available so that the arithmetic stack may grow downwards.

Note that strings must be aligned on the arithmetic stack so that the character count is on a word boundary. All entries on the stack must be even length, so that a string of odd length has one byte at the end which contains no information.

9.11 The Channel Table

A channel number (#n) is an index to an entry in the SuperBASIC channel table. This is a table of items which are each of length **CH.LENCH** (currently $28) bytes. The base of the table is at **BV_CHBAS(A6)**, and the top is at **BV_CHP(A6)**; thus the base of the entry for channel #n is given by:
(n*CH.LENCH+BV_CHBAS(A6))(A6)

The format of each table entry is as follows:

$00	long	the channel ID
$04	float	current graphics cursor (x)
$0A	float	current graphics cursor (y)
$10	float	turtle angle (degrees)
$16	byte	pen status (0 is up, 1 is down)
$20	word	character position on line for PRINT and INPUT
$22	word	WIDTH of page

If a channel entry is off the top of the channel table, or if the channel ID is negative, there is no channel open to that # number.

10.0 Hardware-related Programming

10.1 Memory Map [QL]

The 68008 has one megabyte of address space. Although an unexpanded QL uses only the bottom 256 kbytes of this, the allocation for the remainder is determined and should be adhered to when designing add-on hardware. This is how it is made up:

$FFFFF	
	Add-on ROM (up to 128 kbytes)
$E0000	
	Add-on peripherals (8 slots of up to 16 kbytes each)
$C0000	
	Add-on RAM (up to 512 kbytes)
$40000	
	On-board user RAM (96 kbytes)
$28000	
	Screen RAM (32 kbytes)
$20000	
	On-board I/O (Partially decoded)
$10000	
	Plug-in ROM cartridge (16 kbytes)
$0C000	
	On-board ROM (48 kbytes)
$00000	

The registers in the on-board I/O area are partially decoded: the details of this decode may vary according to different versions of the QL hardware - some versions will recognise any address in the entire area.

However, the address map normally used is the same for all QLs:

Address (hex)	Function (read)	Function (write)
$18023	Microdrive data (track 2)	Display control
$18022	Microdrive data (track 1)	Microdrive/RS232-C data
$18021	Interrupt/IPC link status	Interrupt control
$18020	Microdrive/RS232-C status	Microdrive control
$18003	Real-time clock byte 3	IPC link control
$18002	Real-time clock byte 2	Transmit control
$18001	Real-time clock byte 1	Real-time clock step
$18000	Real-time clock byte 0	Real-time clock reset

The display control registers are in the ZX8301 "Master chip", and the others are in the ZX8302 "Peripheral chip". The details of the QL hardware are rather obscure, and it is strongly recommended that these registers should not be used by applications programs, and should only be accessed via Qdos traps or vectored routines.

For other hardware, e.g. the Miracle Gold card or the QL-Emulator for the ATARI ST, the area from $C0000 is filled up with contigous memory (up to $3FFFFF).

10.2 Display Control

The display format in memory is explained below: this format is specific to the QL and may change on future Sinclair products. It is, therefore, strongly advised that screen output be performed using only the standard screen driver, together with the **SMS.DMOD** trap.

In 512-pixel mode, two bits per pixel are used, and the GREEN and BLUE signals are tied together, giving a choice of four colours: black, white, green and red. On a monochrome screen, this will translate as a four-level greyscale.

In 256-pixel mode, four bits per pixel are used: one bit each for Red, Green and Blue, and one bit for flashing. The flash bit operates as a toggle: when set for the first time, it freezes the background colour at the value set by R, G and B, and starts flashing at the next bit in the line; when set for the second time, it stops flashing. Flashing is always cleared at the beginning of a raster line.

Addressing for display memory starts at the bottom of dynamic RAM and progresses in the order of the raster scan - from left to right and from top to bottom of the picture. Each word in display memory is formatted as follows:

	High byte (A0=0)	Low Byte (A0=1)	
Bit	D7 D6 D5 D4 D3 D2 D1 D0	D7 D6 D5 D4 D3 D2 D1 D0	Mode
	G7 G6 G5 G4 G3 G2 G1 G0	R7 R6 R5 R4 R3 R2 R1 R0	512-pixel
	G3 F3 G2 F2 G1 F1 G0 F0	R3 B3 R2 B2 R1 B1 R0 B0	256-pixel

R, G, B and F in the above refer to Red, Green, Blue and Flash. The numbering is such that a binary word appears written as it will appear on the display: i.e. R0 is the value of Red for the rightmost pixel, that is the last pixel to be shifted out onto the raster.

10.3 Display Control Register

This is a write-only register, which is at $18063 in the QL.

One of its bits is available through the Qdos **SMS.DMOD** trap: bit 3, which is 0 for 512-pixel mode and 1 for 256-pixel mode.

The other two bits of the display control register are not supported by Qdos, these being bit 1 of the display control register, which can be used to blank the display completely, and bit 7, which can be used to switch the base of screen memory from $20000 to $28000. Future versions of Qdos may allow the system variables to be initialised at at $30000 to take advantage of this dual-screen feature: the present version does not.

Bits 0, 2, 4, 5 and 6 of the display control register should never be set to anything other than zero, as they are reserved and may have unpredictable results in future versions of the QL hardware.

10.4 Keyboard and Sound Control

The keyboard and loudspeaker are controlled by the QL's second processor, which is an 8049 single-chip microcomputer: this is known in the QL as the Intelligent Peripheral Controller, or IPC. The **SMS.HDOP** trap provides a set of commands that the CPU can send to the IPC over the serial link that connects them. This trap is discussed in greater detail in section 13.0.

When the keyboard is accessed via the console driver, the usual functions of debounce and conversion to ASCII are performed, in addition to the functions described in section 15.0. The other way of accessing the keyboard is to use the **SMS.HDOP** trap to monitor the instantaneous state of the keys directly: this is the only way of detecting multiple key presses (necessary for joystick input), or of detecting the state of the SHIFT, CTRL and ALT keys when no other key has been depressed. See the SuperBASIC Keywords entry on the **KEYROW** function for an example of the use of this technique.

The same trap, with different parameters, is used for sound generation.

10.5 Serial I/O

The QL's serial I/O should only be accessed via the serial driver, except for setting the baud rate, which is performed by the **SMS.COMM** trap. The only other function that can safely be performed by the user independently of the operating system is the checking of the transmit handshake lines (DTR on channel 1 and CTS on channel 2), which can be looked at by monitoring bits 4 and 5 of the microdrive status register respectively. Note that if the connector is rewired to use these pins as data lines, this function could be used to perform RS232-C reception entirely in software, which would make it possible to perform XON-XOFF handshaking or split baud rate operation.

10.6 Real-time Clock

The QL's real-time clock is a 32-bit seconds counter. The three traps **SMS.RRTC, SMS.SRTC** and **SMS.ARTC** are used to read, set and adjust the clock. The vectored routines **CV.ILDAT** and **CV.ILDAY** are used to convert the time obtained to a string.

10.7 Network

This should not be accessed other than by the built-in device driver.

10.8 Microdrives

Normally, these should not be accessed other than by the built-in device driver. However, it is possible to write routines to access microdrive sectors directly in order to perform such functions as fast medium-to-medium copying or recovery of data from a damaged medium.

There are four vectored routines provided for this purpose: **MD.READ, MD.WRITE, MD.VERIF** and **MD.RDHDR**. Use of these routines requires a detailed understanding of the microdrive hardware and format, and is probably beyond the scope of most users.

However, to use these routines the following code example shows how a microdrive is selected or de-selected. In later versions of the operating system it will be a vectored entry.

```
sys_wser
     move.b   d0,-(sp)                    ; save operation
wait
     subq.w   #1,sys_tmot(a0)             ; decrement timeout
     blt.s    set_mode                    ; done?
     move.w   #(20000*15-82)/36,d0        ; time=18*n+42 cycles
delay1
     dbra     d0,delay1                   ; delay
     bra.s    wait                        ; repeat until timeout expires
set_mode
     clr.w    sys_tmot(a0)                ; clear wait
     and.b    #pc.notmd,sys_tmod(a0)      ; not RS232
     move.b   (sp)+,d0
     or.b     d0,sys_tmod(a0)             ; either mdv or net
     and.b    #$ff-pc.maskt,sys_qlir(a0)  ; disable transmit interrupt
exit
     move.b   sys_tmod(a0),pc_tctrl       ; set PC
     rts
sys_rser
     bclr     #pc..serb,sys_tmod(a0)      ; set RS232 mode
     or.b     #pc.maskt,sys_qlir(a0)      ; enable transmit intertupt
     bra.s    exit
md_desel
     moveq    #pc.desel,d2                ; clock in deselect bit first
     moveq    #7,d1                       ; deselect all
     bra.s    sedes
md_selec
     moveq    #pc.selec,d2                ; clock in select bit first
     subq.w   #1,d1                       ; and clock it through n times
sedes
clk_loop
     move.b   d2,(a3)                     ; clock high
     moveq    #(18*15-40)/4,d0            ; time=2*n+20 cycles
     ror.l    d0,d0
     bclr     #pc..sclk,d2                ; clock low
     move.b   d2,(a3)                     ;  ...  clocks  d2.0  into  first
drive
     moveq    #(18*15-40)/4,d0            ; time=2*n+20 cycles
     ror.l    d0,d0
     moveq    #pc.desel,d2                ; clock high - deselect bit next
     dbra     d1,clk_loop
     rts
drive
     bsr.s    startup
     bsr.s    wind_dwn
     rts
```

```
; Routine to start up a microdrive
; RETURNS IN SUPERVISOR MODE (if D3=1 to 8)
;
;        Entry                   Exit
;    D1                          D1   smashed
;    D2                          D2   smashed
;    D3.L number of microdrive   D3   preserved
;    A0                          A0   SYS_BASE
;    A3                          A3   mdctrl (=$18020)
;
;    Error returns:
;         orng microdrive out of range
startup
     cmp.l     #1,d3                       ; legal microdrive?
     blt.s     ill_drve                    ; jump if not
     cmp.w     #8,d3                       ; legal microdrive?
     bgt.s     ill_drve                    ; jump if not
     move.l    (sp)+,a3                    ; A3=return address
     moveq     #sms.info,d0                ; get system variables
     trap      #do.sms2                    ; get system variables
     trap      #0                          ; supervisor mode
     move.l    a3,-(sp)                    ; 'return' the return address
     moveq     #$10,d0                     ; microdrive mode
     bsr       sys_wser                    ; wait for RS232 to complete
     or.w      #$0700,sr                   ; shut out rest of world
     move.l    d3,d1                       ; d1 is microdrive to be started
     move.l    #pc_mctrl,a3                ; control register
     bsr       md_selec                    ; start it up
     moveq     #0,d0                       ; no problems
     rts                                   ; return
ill_drve
     moveq     #err.orng,d0                ; error!
     rts

; Routine to wind down (all!!!) microdrives
; MUST BE CALLED IN SUPERVISOR MODE
;
;        Entry                   Exit
;    D1                          D1   smashed
;    D2                          D2   smashed
;    A0                          A0   SYS_BASE
;    A3                          A3   ptr to instruction after call to here
wind_dwn
     moveq     #sms.info,d0
     trap      #do.sms2                    ; get system variables
     move.l    #pc.mctrl,a3                ; control register
     bsr.s     md_desel                    ; wind it down
     bsr       sys_rser                    ; re-enable RS232
     move.l    (sp)+,a3                    ; A3=return address
     move.w    #0,sr                       ; enable interrupts, exit SV-mode
     move.l    a3,-(sp)                    ; return address
     rts                                   ; return
```

10.10 User and Supervisor Mode [ST]

Motorola has implemented function code lines into their processors to allow for hardware memory protection. This has never been used on a QL, and for the first two QL-Emulators for the ATARI's the machines had to be modified to ignore the function code line which says whether an access is done in supervisor mode or user mode - the hardware always thought the access is in supervisor mode. Generally, allowing accesses to the system addresses in supervisor mode only is a good idea. This traps a program which tries to destroy some vectors or modify the hardware settings by mistake or due to a programming fault.

Accesses to the system vectors ($000 to $400) have to be done in supervisor mode, otherwise the system will generate a bus error. The only execption is an access to a QL utility vector which may be accessed in both modes, e.g.

```
     MOVE.W    RI.EXEC,A2
     JSR       (A2)
```

Hardware registers should be modified by the supervisor only, therefore any access to ST hardware registers ($FFxxxxxx to $FFFFFFFF) are allowed in supervisor mode only - no exception! Again, doing it in user mode results in a bus error. The same applies for accesses to non-existent hardware - a bus error is generated. In general there should be no need to access non-existent hardware, as the facilities of the system can be discovered by looking at system variables or the thing list, if a thing does not exist, then the hardware is simply not available on this machine. If a hardware address has to be accessed and it is not known whether the machine supports it or not, the following routine could be used to do it.

```
; Call routine with own bus error handler    ©1992 Jochen Merz
; Call a user-supplied routine to access hardware addresses
; and ignore internal bus error handler to find out if routine succeeds.
; This routine must be called in supervisor mode!
; The routine which is to be called must not modify d3-d4 and a3, but
; it should reset d0 on success or return any other error!
;
;         Entry                   Exit
;
;    D1   call parameter          return parameter
;    D2   call parameter          return parameter
;    D3+                          preserved
;
;    A0   routine to be called    return parameter
;    A1   call parameter          return parameter
;    A2+                          preserved
;
;    Error returns:ERR.NIMP if bus error occured
;                  any error returned by supplied routine
;---
cbus_reg reg d3-d4/a3-a4
ut_cbuser
     movem.l   cbus_reg,-(sp)
     move.w    sr,d3                   ; keep SR
     or.w      #$0700,sr               ; no interrupts allowed
     move.l    sp,a3                   ; keep SSP
     move.l    $0008,d4                ; get standard bus error
     lea       buserr,a4
     move.l    a4,$0008                ; and insert new one
     moveq     #err.nimp,d0            ; assume bus error
     jsr       (a0)                    ; call routine
buserr
```

```
move.l    a3,sp              ; restore stack
move.l    d4,$0008           ; restore bus error
move.w    d3,sr              ; restore SR
movem.l   (sp)+,cbus_reg
tst.l     d0
rts
```

The routine at (A0) should first access the hardware register which is to be tested. If this fails, the routine is left immediately. If not, it can do whatever it wants and return with an RTS.

10.11 The Interrupt System [ST]

All I/O on the ATARI is done under interrupt. This means, disabling the interrupts for a longer period of time should be avoided. At present, there are two different interrupt systems implemented: one for the old ST models, which uses the VBLANK interrupt for calling the Poll loop. The disadvantage is, that it is unknown whether the poll is called at 50, 60 or even 71 Hz, because this depends on the monitor which is connected.

On STE and TT models the poll is a steady 50 Hz interrupt, not related to the VBLANK. It is derived from a 200 Hz interrupt which generates a software level 1 interrupt.

The general rules are: try to avoid disabling the interrupts at all. If you have to, don't stay long in this mode (Sometimes you have to, e.g. for accesses to the sound chip - there must be no interrupt between register select and register read/write)! Never modify the interrupt system! Do not modify the masks in the SCU!

If you need a timer, the system may provide a timer. Check for a thing named "Timer" by trying to use it. If it is in use, someone else is using the timer. If it is not found, the timer is not available at all. If it is successful (it should be, generally spoken) then the Timer B of the MFP is your's. The Thing itself does nothing but making sure that only one job can use the timer at a time, and it also disables the interrupt on force remove. The server routine for the timer interrupt has to be inserted at $1A0. The timer can be programmed to any rate which is possible, but you should refer to other documentation which gives detailed description of the MFP.

10.12 The MIDI Interrupt server [ST]

The MIDI interrupt server is invoked through the keyboard server. To locate the keyboard server, scan through the polling linked list looking for 'ASTK' iod_pllk (8) bytes below the polling link (i.e. the base of a standard linkage block). Then put the base address of the midi linkage at $a8 in the keyboard linkage and the address of the MIDI server at $ac.

The MIDI server is called with A3 pointing to the MIDI linkage and D0.b holding the contents of the MIDI status register. (D0.b will always be negative - i.e. the interrupt bit will be set.) The server may smash D0/D1/A0/A2/A3 and should return with RTE. Due to an error in old keyboard drivers, A3 is not saved on a MIDI call. This means, that when you look for the 'ASTK' flag, this address should be kept and A3 should be set to this linkage address just before the MIDI server returns with RTE.

10.13 Different Processors [ST]

You can find out which processor is running the system by having a look at the system variable SYS_PTYP ($A1). The high nibble contains the processor type, which gives a byte value of $0x for a 68000, $1x for a 68010, $2x, $3x and $4x for 68020, 68030 and 68040, respectively. It is a good idea to write a branch by looking at this register for time-critical routines which could be improved by using the extended 68020+ register set.

The low nibble is reserved to show the presence of MMUs and Floating Point Coprocessors. It is, at present, usually 0.

The different processors differ a bit in user-mode handling of some instructions. QDOS programs had a number of privilege violation problems, but these are emulated now. The most common problem is the entry to Supervisor mode, which is usually something like

```
move.w   SR,Dx          ; save previous processor mode
trap     #0             ; into supervisor mode
     ... supervisor mode code
move.w   Dx,SR          ; back to previous mode
```

Processors other than 68000s will generate a Privilege Violation exception on the first command, as it is not allowed to read the status register in user mode! Therefore, all reads of the status register are emulated. As all the other privilege violation cases will definitely lead to a program malfunction, the programm loops in an endless loop, waiting to be removed from the system. If you set a debugger on this program and display the memory after the PC, then you will see a message "Priv V at (A0). The offending instruction can be found at the address to which A0 points.

10.14 Different Machines [ST, SMSQ]

It might be very helpful to know on which machine the current programs are running. They all differ in hardware, and behave different in some ways. The standard application usually does not need to know on which machine it is running, but it could be very useful for some special applications to use hardware if it exists to speed up things on some machines. In addition, it could be helpful to know which type of emulator is installed in the machine. The system variable SYS_MTYP ($A7) gives details about the machine. At present, the definition is as follows: Bits 4 to 0 contains the machine type, bits 7 to 5 the display type:

0	for all ordinary ST's without realtime-clock.
2	for Mega ST or ST's with realtime clock.
4	for Stacy.
6	for ordinary STE.
8	for Mega STE.
10	for GoldCard.
12	for SuperGoldCard
16	for the Falcon 030.
24	for the TT.
28	for the QXL

In addition, bit 0 is set if the machine contains a Blitter chip (ATARIs only) or a Hermes (QL).

The display types are:
%000 for the Futura emulator (we cannot tell whether it gives real MODE8 or not), %010 for the Extended 4 Emulator and %100 for the QVME emulator card. %001 stands for ATARI monochrome mode, and %110 for VGA mode (e.g. QXL).

Please note that this system variable is supported from E.20 onwards, together with E-Init software V1.07 or later. If this system variable is 0 you can assume a normal ST with an old emulator or, which is more likely, old software.

10.15 The ATARI DMA [ST]

The DMA is used to handle the floppy disk system and the ACSI port. You may gain access to the DMA by trying to TAS the system variable SYS_DMIU ($A6). If this is set, you may use the DMA (e.g. to provide new device drivers for streamers or CD ROMs connected to the ACSI port). You should clear this flag as soon as possible.

As SMSQ supports more than one type of RAM, a key has been added to allow for the controlled allocation of specific RAM. The ATARI TT may have Fast RAM in addition of the standard ST compatible RAM. This Fast RAM cannot be used for Floppy Disk DMA and DMA from and to devices connected to the ACSI port (this includes the ATARI LaserPrinter SLM 804 and SLM 605). It is possible to pass the characters "ACSI" in D3 on the SMS.ACHP call to make sure that only the type of RAM is allocated wich supports direct memory access to the ACSI port.

11.0 Adding Peripheral Cards to the QL

Peripheral cards may be plugged into the expansion connector on the left-hand side of the QL.

There are two general categories of peripheral card for the QL: pure add-on memory cards, and other peripheral cards.

It is intended that only one pure add-on RAM pard be plugged into the machine at any one time. It is allocated the address area between $40000 and $BFFFF; the add-on memory should be contiguous from $40000 upwards. This allows for an add-on memory size of up to 512 kbytes.

There is also room for an add-on ROM card of up to 128kbytes, which is allocated the addresses $E0000 to $FFFFF.

Other peripheral cards contain electronics for the devices being added, a small ROM containing the drivers for the devices being added together with a code allowing the QL to detect that the card is present, and a 4-bit comparator which is used to select the card as explained below.

Note that the convention adopted in this document for an active low signal is to append the letter "L" to the end of the signal name, as in DTACKL, VPAL etc. This takes the place of the overbar indication used in the data sheets from most vendors.

11.1 Expansion Connector

The expansion connector allows extra peripherals to be plugged into the QL. Details of the connections available at the connector may be found in the QL Concepts manual.

The connector inside the QL is a 64-way male DIN-41612 indirect edge connector, as found on standard Eurocard modules. The connector on each add-on card should be the inverse version of this.

The VIN supply is in the region of +9V DC: the trough never falling below 7V. Up to 500 mA may be drawn from this to power the card.

No add-on card should load any pin on the edge connector by more than two LSTTL loads. All add-on card data bus output drivers should be a 74LS245 or equivalent, in terms of drive ability, and being tri-state.

11.2 CPU Interface

The CPU interface is totally memory-mapped onto the 68008's bus, control of the bus for use with the video display controller being obtained by using the DTACKL signal to arbitrate the bus. Memory access is entirely controlled by DSL, with ASL left unused. ASL should not be used to gate any add-on hardware.

An unexpanded QL does not look at address lines A19 and A18. In peripheral cards which are to be added to the QL, it is necessary for each card to disable the circuitry on the QL itself when that peripheral card recognises its own address. This is achieved by pulling signal DSMCL high before DSL goes low including buffering times. This is done typically by using a fast NPN switching transistor (such as an MPS2369) connected as an emitter follower with the emitter connected to DSMCL, the collector to +5V and the base to a logic signal. Note that the timing for this operation is the most critical in most hardware interfaces to the QL, especially when the necessary signals have been buffered.

Add-on cards must supply DTACKL or VPAL as required, to notify the CPU that they have recognised their address.

All 68008 signals are available on the expansion connector to allow expansion to include coprocessors or other peripherals.

The following signals are outputs only: A0-A19, RDWL, ASL, DSL, BGL, CLKCPU, E, RED, BLUE, GREEN, CSYNCL, ,VSYNCH, ROMOEH, FC0-2, RESETCPUL.

The following lines are inputs only, and should only be driven from open collector outputs: DTACKL, BRL, VPAL, IPL0L, IPL1L, BERRL, EXTINTL, DBGL.

The data bus, D0-D7, is bidirectional.

The EXTINTL pin may be used to generate a level 2 external interrupt, which can be linked to a user task (see section 6.3). Note that the EXTINTL pin must not be negated until the Qdos start-up mechanism is complete, or there is a risk of the system hanging up.

11.3 Peripheral Card Addressing

Peripheral cards (other than pure add-on memory cards) are allocated the address space between $C0000 and $DFFFF. Each peripheral card, when selected, must disable DSMCL and assert VPAL or DTACKL as required, for its own use. This address pace is split into eight slots of 16kbytes each; each peripheral card should normally take only one block if a full set of eight peripheral cards is to be allowed to operate concurrently.

There is a set of four select lines, SP0-SP3, appearing on the edge connector. The first card in an expansion module, or a single card directly plugged into the QL, receives a value of zero on these for lines. Each slot in an expansion module has a value one different from that in the other slots: this means that each card is allocated 16kbytes of address space. The card select logic compares the values on A17-A14 against the number coming in on the select lines in order to determine whether that card is selected. For the card to be selected it must be the case that A14=SP0, A15=SP1, A16=SP2 and A17=SP3.

If there is a ROM containing device drivers for the peripheral card, it should sit in the bottom addresses of the 16kbyte block. The format of the lowest part of this ROM is specified in the next section.

11.4 Add-on Card ROMs

When the machine is booted, the operating system checks for plug-in ROM drivers by looking for the characteristic longword flag $4AFB0001 at the base of each location in which a ROM might be present. The beginning of a plug-in ROM should be in the following format:

00	$4AFB0001 (flag to indicate ROM is present)
04	pointer to list of BASIC procedures and functions
06	pointer to initialisation routine
08	string identifying the ROM

The pointers are relative to the base of the ROM. If the list pointer is zero then there will be no attempt to link routines into SuperBASIC.

The list of BASIC procedures and functions is in the form used by **SB.INIPR** (see section 16.0).

At start-up the machine will link in the additional BASIC procedures from the ROM, then call the initialisation routine (in user mode) which must not modify A6, and finally must restore A0 (the initial window ID), and A3, the pointer to the ROM, on exit. Up to 128 bytes may be used on the user stack.

The description should be in the form of a character count (word) followed by the ASCII characters of the device description(s) ending with the newline character (ASCII 10). It is recommended that the number of characters should be limited to 36.

All code for device drivers must be position independent, since the addresses of the ROM and the devices on the card will be dependent upon the position at which it has been plugged into a QL expansion module. This allows multiple copies of the same add-on card to be used simultaneously.

12.0 Non-English Systems

There are three areas in which non-English QLs may differ from English QLs: the video, the keyboard, and the character set for serial communications.

The version codes for non-English QLs are adjusted appropriately to contain a character identifying the country. In the version code returned by **SMS.INFO**, this character replaces the decimal point; in the string returned by the SuperBASIC **VER$** function, the character is added on at the end, producing a string three characters long for non-English QLs. Example:

1G13 MGG

12.1 Video

This is different for countries where the television system is NTSC, which permits the use of fewer raster lines than PAL. In QLs for such countries, the following options are the defaults:

For monitor operation, a 50Hz 624-line non-interlaced system is used; this is the same system as is used on the English QL. The full 512x256 pixel display is available, and the default windows and character size are the same as for the monitor mode on an English QL.

For TV operation, a 60Hz 524-line non-interlaced system is used in which the number of raster lines available is limited to 192. In order to ease the task of software conversion, an alternate display font is provided which allows a 6x8 character square instead of the usual 6x10. This ensures approximately the same number of visible rows of text on both PAL and NTSC QLs, at the cost of true descenders and reduced vertical spacing. The default windows and graphics scaling for TV operation are different from those of the English QL.

12.2 Non-English-language Keyboards

The keyboard layout for most European countries will be different from the English layout. This difference should be largely transparent to applications software, since the "QL ASCII" codes contain all the characters necessary for the European countries in question, and the codes generated are independent of the keyboard layout and hence of the actual key depressions required to generate them.

However, there are a few subtleties, the following being the most obvious:

1. A program which draws pictures of keys in certain places will certainly produce an incorrect drawing if the location of those keys has changed between countries.

2. The keyrow function (or **SMS.HDOP** trap) refers to the physical position of the keys, not to their logical meaning. For example, a test on an English QL for the letter "Q" using keyrow will turn into a test for the letter "A" on a French QL which has an AZERTY keyboard.

3. An instruction to "hit any key" will not be strictly accurate for a country which employs non-spacing diacriticals, where the keypress of an accent character does not generate a code until the character to be accented is pressed. The length of the type-ahead buffer in the IPC will be apparently reduced in such cases.

12.3 Character Set [not SMS2] [SMSQ]

The English character set is available in all countries. However, in non-English countries, the character set for serial communications may (optionally) be translated into a "local" character set. A further option allows the user to specify his own translation table, since it is anticipated that a number of countries will have several standards (i.e., no standards at all).

The trap **SMS.TRNS** is used to set up user-supplied translation tables for the serial communications (serial and parallel printer ports). In addition, a language-dependant table for the error-messages may be supplied.

The simple translation exchanges a character code against another one. The character may optionally be replaced by three characters, using a second table.

The format of the translation table is as follows:

```
base_of_table
          word      $4AFB                       flag
          word      table1-base_of_table        relative pointer to first table
          word      table2-base_of_table        relative pointer to second table
table1
          256 bytes                             1 to 1 character translation
table2
          byte                                  number of translations or 0
          for every translation ...:
          byte                                  character to be translated
          3 bytes                               three replacement characters
```

If the first pointer is zero, no translation is being performed.
The second table is only used for output.

The message table, which may be optionally supplied, has to be in the following format:

```
base
          word      $4AFB               flag
          word      err_nc-base         rel. pointer to 'not-complete' message
          word      err_ijob-base       rel. pointer to 'invalid job' message
          ...
          ...                           all error messages
          ...
          word      err_isyn-base       rel. pointer to 'bad line' message
          word      atline-base *       message 'At line '
          word      sectors-base        message ' sectors'
          word      F1_F2-base          message 'F1 .. monitor'
                                                'F2 .. TV'
          word      copyright-base *    message 'C1983 Sinclair Research Ltd'
          word      dur_when-base       message 'during WHEN processing'
          word      procclr-base        message 'PROC/FN cleared'
          word      days-base *         days 'SunMonTueWedThuFriSat'
          word      months-base *       months 'JanFebMar ..' etc
```

All messages except the days and months have to be in standard string format.
All messages except those marked with * should end with newline (ASCII 10).

12.4 Special Alphabets

Languages with non-Roman alphabets, such as Hebrew, Greek, Thai, Arabic, etc., require special treatment. No general scheme has been devised for making software transportable to these countries, and the implementation means will be specific to each country.

13.0 System Traps

Trap #1 D0=$18 — SMS.ACHP

Allocate common heap area

Call parameters		Return parameters	
D1.L	number of bytes required	D1.L	nr. of bytes allocated
D2.L	owner job ID	D2	???
D3	0 or "ACSI"	D3	???
		D4+	all preserved
A0		A0	base address of area
A1		A1	???
A2		A2	???
A3		A3	???
		A4+	all preserved

Error returns (Z flag is not always set correctly):

IMEM out of memory
IJOB job does not exist

This trap is a specific example of the general heap allocation mechanism described in section 4.1 and accessible using **SMS.ALHP**.
ATARI TT (or similar machines with ST RAM and Fast RAM) only: If D3 is passed as "ACSI", then memory is allocated in ST compatible RAM, not in Fast RAM [SMSQ].

Trap #1 D0=$A — SMS.ACJB

Activate a job

Call parameters		Return parameters	
D1.L	job ID	D1.L	job ID
D2.B	priority	D2	preserved
D3	timeout (0 or -1)	D3	preserved
		D4+	all preserved
A0		A0	base of job ctrl area
A1		A1	preserved
A2		A2	preserved
A3		A3	preserved if D3=0
		A4+	all preserved

Error returns:

IJOB job does not exist
NC job already active

This trap activates a job in the transient area. Execution commences at the start address defined when the job was created.

If the timeout is zero then the execution of the current job continues, otherwise the current job will be suspended until the job activated is completed. The trapp will then return with the error code (if any) from that job.

Trap #1 D0=$C — SMS.ALHP

Allocate an area in a heap

Call parameters		Return parameters	
D1.L	length required	D1.L	length allocated
D2		D2	???
D3		D3	???
		D4+	all preserved
A0	ptr to ptr to free space	A0	base of area allocated
A1		A1	???
A2		A2	???
A3		A3	???
A6	base address	A6	preserved

Error returns:

IMEM no free space large enough

Two trap entries are provided for user heap management where this is required to be atomic. A6 is used as a base address for both this call and for **SMS.REHP** so that A0 (and A1) is an address relative to A6.

See section 2.1.4 for details of the heap mechanism.

Trap #1 D0=$16 — SMS.AMPA

Allocate BASIC program area

Call parameters		Return parameters	
D1.L	number of bytes required	D1.L	nr. of bytes allocated
D2		D2	???
D3		D3	???
		D4+	all preserved
A0		A0	???
A1		A1	???
A2		A2	???
A3		A3	???
A6	base address	A6	new base address
A7	user stack pointer	A7	new stack pointer

Error returns:

IMEM out of memory

Trap #1 D0=$E

SMS.ARPA

Allocate resident procedure area

Call parameters		Return parameters	
D1.L	number of bytes required	D1	???
D2		D2	???
D3		D3	???
		D4+	all preserved
A0		A0	base address of area
A1		A1	???
A2		A2	???
A3		A3	???
		A4+	all preserved

Error returns:

IMEM out of memory
NC unable to allocaste (TRNSP area not empty)

This trap should only be invoked when the transient program area is empty.

Trap #1 D0=$15

SMS.ARTC

Adjust real-time-clock

Call parameters		Return parameters	
D1.L	adjustment in seconds	D1.L	time in seconds
D2		D2	???
D3		D3	???
		D4+	all preserved
A0		A0	???
A1		A1	preserved
A2		A2	preserved
		A3+	all preserved

As setting the clock takes a significant time, no adjustment is made if a call is made to adjust the clock and D1=0.

Time starts at 00:00:00, 1. January 1961.

Trap #1 D0=$2F [SMSQ] SMS.CACH

Turn Cache on or off

Call parameters		Return parameters	
D1.L	1 for Cache on, 0 for Cache off, -1 to read current cache setting	D1	1 = Cache on, 0 = Cache off

Error returns:

always okay

No other value than 0 or 1 should be used to set the cache, to allow for future cache control strategies. To read the current cache setting, use -1. For Motorola 68000 processors, it always returns 0.

Trap #1 D0=$12 SMS.COMM

Set the baud rate

Call parameters		Return parameters	
D1.W	baud rate	D1	???
D2		D2	preserved
D3		D3	preserved
		D4+	all preserved
A0		A0	preserved
A1		A1	preserved
A2		A2	preserved
A3		A3	preserved
		A4+	all preserved

Error returns:

IPAR non recognised baud rate

For a standard QL, the baud rate supplied in D1 is applied to both serial ports. For extended Systems (e.g. Hermes) refer to the specific documentation supplied with the extension.

Trap #1 D0=$1

SMS.CRJB

Create a job in transient program area

Call parameters		Return parameters	
D1.L	owner job ID	D1.L	job ID
D2.L	length of code (bytes)	D2	preserved
D3.L	length of data space	D3	preserved
		D4+	all preserved
A0		A0	base of area allocated
A1	start address or 0	A1	preserved
A2		A2	preserved
A3		A3	preserved
		A4+	all preserved

Error returns:

IMEM out of memory
IJOB no room in job table or D1 is not a job

This trap allocates space in the transient program area, and sets up a job entry in the scheduler tables. This does not invoke the job and the only initialisation is that two words of 0 are put on the stack. The program itself would normally be loaded, by another job, into the space allocated, after this system call. The stack pointer saved in the job control area points to two zero words on the stack (at the highest addresses in the job's data area); if channels are to be opened for the job, or a command string is to be passed to the job, then this can be done before the job is activated.
If D1 is 0 (i.e. owned by the system), the new job is independent, if D1 is negative, it is owned by the calling job.

Trap #1 D0=$10 — SMS.DMOD

Set or read the display mode

Call parameters			Return parameters	
D1.B	key:	-1 read mode	D1.B	display mode
		0 mode is 4 colour		
		2 mode is 2 colour [SMS]		
		8 mode is 8 colour		
		12 mode is 16 colour [Thor XVI]		
D2.B	key:	-1 read display	D2.B	display type
		0 monitor		
		1 625-line TV		
		2 525-line TV		
D3			D3	preserved
			D4+	all preserved
A0			A0	???
A1			A1	preserved
A2			A2	preserved
A3			A3	preserved
			A4	???

This call is used to set or read the current display mode. It is treated as a manager trap as it affects all the displayed windows. If a call is made to set the screen mode, then all the windows on the screen are cleared and the character sizes may be adjusted. Oviously, there are serious risks involved in calling this trap to set the mode when there are jobs in the machine accessing the screen.

For a SMS machine or Extended4-Emulator, this trap only clears the windows of the calling job, so that the windows of other jobs are not affected.

Trap #1 D0=$7 — SMS.EXV

Set the per-job pointer to trap vectors

Call parameters		Return parameters	
D1.L	job ID	D1.L	job ID
D2		D2	preserved
D3		D3	preserved
		D4+	all preserved
A0		A0	base of job
A1	pointer to table	A1	???
A2		A2	preserved
A3		A3	preserved
		A4+	all preserved

Note: When a routine in the table is entered as a result of an exception, the CPU is in supervisor mode. The routine should return with an RTE command (not RTS). Any registers used must be saved and restored.

Trap #1 D0=$35 [SMSQ] SMS.FPRM

Find Preferred Module

For details on this trap call, refer to section 19, "Language dependent Modules".

Trap #1 D0=$5 SMS.FRJB

Force-remove job from transient program area

Call parameters		Return parameters	
D1.L	job ID	D1	???
D2		D2	???
D3.L	error code	D3	???
		D4+	all preserved
A0		A0	???
A1		A1	???
A2		A2	???
A3		A3	???
		A4+	all preserved

Error returns:

IJOB job does not exist

This trap inactivates a complete job tree and deletes all jobs in it. If D1 is set to -1 then the current job is removed.

Neither of the traps **SMS.FRJB** or **SMS.RMJB** to remove jobs can remove job 0.
Neither of these traps are guaranteed to be atomic.

If there is a job waiting on completion of any job removed, this is released with D0 set to the error code (see **SMS.ACJB** D0=$A).

Trap #1 D0=$6 SMS.FRTP

Find largest contigous free space that may be allocated in transient prog area

Call parameters		Return parameters	
D1		D1.L	length of space found
D2		D2	???
D3		D3	???
		D4+	all preserved
A0		A0	???
A1		A1	???
A2		A2	???
A3		A3	???
		A4+	all preserved

Trap #1 D0=$29 [SMS2] [EXT] SMS.FTHG

Free Thing

Call parameters		Return parameters	
D1	user Job ID	D1	preserved
D2	parameter	D2	returned result
D3	parameter	D3	preserved
		D4+	all preserved
A0	name of Thing to free	A0	preserved
A1	parameter	A1	???
A2	parameter	A2	returned result
		A3+	all preserved

Error returns:

ITNF Thing was not found
any returns from Thing's FREE code

This routine will usually be called when a Job no longer requires the use of a Thing. If a Thing is freed on behalf of a Job other than the calling Job, then the user Job is removed, as it would probably otherwise continue trying to use the Thing. As with the call to use a Thing, additional parameters may be required or returned by the Thing itself.

Trap #1 D0=$11 — SMS.HDOP

Send a command to the IPC

Call parameters		Return parameters	
D1		D1.B	return parameter
D2		D2.L	preserved
D3		D3	preserved
		D5	???
		D7	???
A0		A0	preserved
A1		A1	preserved
A2		A2	preserved
A3	pointer to command	A3	preserved
		A4+	all preserved

This trap sends a command to the IPC.

A command sent to the IPC is a nibble (4 bits of a byte) followed by a stream of nibbles or bytes being the parameters of the command; some information may then be returned from the IPC. The command format for **SMS.HDOP** is a header describing the command to be sent, followed by the parameters to be sent, followed by a byte indicating whether a reply is expected. The IPC communication is completely unprotected and the command must not contain any errors or else the entire machine will hang up. IPC communications is a very slow process and excessive use of the IPC, for example: polling all rows of the keyboard - the cursor keys have been organised to all be in one row, will cause very high processor overheads.

The command format allows 0, 4 or 8 bits to be transferred from each byte in the parameter block. This is encoded in 2 bits:

00	send least significant 4 bits
01	send nothing
10	send all 8 bits
11	send nothing.

The complete command format is:

1 byte	the IPC command nibble in the LS 4 bits;
1 byte	the number of parameter bytes to follow;
1 long word	containing the codes for the amount of each parameter byte to be sent in reverse order: bits 1,0 the amount of first byte to send, bits 3,2 the amount of the second byte etc.;
n bytes	the parameter bytes
1 byte	length of reply encoded in bits 1,0.

Most of the IPC commands are for use by the operating system and any attempt by application programs to use these is liable to cause loss of data or worse. There are three commands for the IPC which may be used by applications programs:

$9 read a row of the keyboard, 1 parameter
- 4 bits the row number
- 8 bits reply

$A initiate sound, 8 parameters
- 8 bits pitch1
- 8 bits pitch2
- 16 bits interval between steps
- 16 bits duration
- 8 bits top 4 bits: step in pitch, lower 4 bits: wrap
- 8 bits top 4 bits: randomness of step, lower 4 bits: fuzziness
- no reply

$B kill sound, no parameters, no reply.

An example of initiate sound is the following line, which is the data for a sirene-type sound:

```
sirene   dc.b $a   ; command nibble
         dc.b 8    ; number of parameter bytes
         dc.l $0000aaaa ; paramters all 8 bit
         dc.b $01,$14,$c8,$00,$ff,$7f,$10,0    ; parameters
         dc.b 1    ; no reply
```

This is equivalent to the SuperBASIC command

BEEP HEX('7FFF'),1,HEX('14'),HEX('00C8'),1,0,0,0

SMS.INFO

Trap #1 D0=$0

System information

Call parameters	Return parameters	
D1	D1.L	current job ID
D2	D2.L	ASCII OS version (n.nn)
D3	D3	preserved
	D4+	all preserved
A0	A0	pointer to system vars
A1	A1	preserved
A2	A2	preserved
A3	A3	preserved
	A4+	all preserved

This trap should always be used as a means of obtaining the base address of the system variables as well as ensuring that the operating system version supports the features you wish to use.

Trap #1 D0=$2 — SMS.INJB

Information on a job

Call parameters		Return parameters	
D1.L	job ID	D1.L	next job in tree
D2.L	job at top of tree	D2.L	owner job
D3		D3.L	MSB -ve if suspended LSB priority
		D4+	all preserved
A0		A0	base address of job
A1		A1	???
A2		A2	preserved
A3		A3	preserved
		A4+	all preserved

Error returns:

IJOB job does not exist

This trap returns the status of a job.

This trap may be used to check the status of a tree of jobs. On each call D2 should be the ID of the job at the top of the tree; to scan a complete tree the trap is made with D1 being the return value of the previous call. When the tree has been completely scanned D1 is returned equal to zero.

Trap #1 D0=$2E — [SMSQ] SMS.IOPR

Set IO Priority

Call parameters		Return parameters	
D1		D1	preserved
D2.W	priority to set	D2	preserved

Error returns:

always okay

The IO priority sets the priority of the IO retry operations. In effect, this sets a limit on the time spent by the scheduler retrying IO operations. A priority of one sets the IO retry scheduling policy to the same as QDOS, thus giving a similar level of response but with a higher crude performance. A priority of 2 will give QDOS levels of response, better response under load. 10, for example, will give a much better response under load but degraded performance. 32767 will give maximum response, the performance depends on the number of jobs waiting for input (default SMSQ setting).

Trap #1 **D0=$31** [SMSQ] SMS.LENQ

Language Enquiry

For details on this trap call, refer to section 19, "Language dependent Modules".

Trap #1 **D0=$1A** SMS.LEXI
D0=$1C **SMS.LPOL**
D0=$1E **SMS.LSHD**
D0=$20 **SMS.LIOD**
D0=$22 **SMS.LFSD**

$1A Link an external interrupt service routine
$1C Link a polling 50/60 Hz service routine
$1E Link a scheduler loop task
$20 Link an IO device driver
$22 Link a directory device driver into the operating system

Call parameters		Return parameters	
D1		D1	preserved
D2		D2	preserved
D3		D3	preserved
		D4+	all preserved
A0	address of link	A0	preserved
A1		A1	???
A2		A2	preserved
A3		A3	preserved
A6		A6	preserved

Trap #1 **D0=$30** [SMSQ] SMS.LLDM

Link in Language Dependent Module

For details on this trap call, refer to section 19, "Language dependent Modules".

Trap #1 **D0=$32** [SMSQ] SMS.LSET

Language Set

For details on this trap call, refer to section 19, "Language dependent Modules".

Trap #1 D0=$26 [SMS2] [EXT] SMS.LTHG

Link in new Thing

Call parameters		Return parameters	
D1		D1	preserved
D2		D2	preserved
D3		D3	preserved
		D4+	all preserved
A0		A0	preserved
A1	address of Thing linkage	A1	preserved
A2		A2	preserved
		A3+	all preserved

Error returns:

FEX Thing of this name already exists

The linkage block should have **TH_THING, TH_USE, TH_FREE, TH_FFREE, TH_REMOV, TH_VERID, TH_SHARE** and **TH_NAME** filled in before this call is made: it must be allocated in the common heap so that **SMS.ZTHG**, or **SMS.RTHG** called from another program, can de-allocate the linkage block correctly. The name in the linkage block is set to lower case, to speed searching.

Trap #1 D0=$34 [SMSQ] SMS.MPTR

Find Message Pointer

For details on this trap call, refer to section 19, "Language dependent Modules".

Trap #1 D0=$2B [SMS2] [EXT] SMS.NTHG

Next Thing

Call parameters		Return parameters	
D1		D1	preserved
D2		D2	preserved
D3		D3	preserved
		D4+	all preserved
A0	thing name or 0	A0	preserved
A1		A1	next thing linkage
A2		A2	preserved
		A3+	all preserved

Error returns:

ITNF Thing was not found

This routine allows code to scan the Thing list to find out what Things are available. On each call the address of the next thing linkage block in the list is returned. If a zero pointer to a thing name is passed then the first block in the list will be returned. The following code will thus scan the entire Thing list:

```
     SUB.L     A0,A0          ; start of list
SLOOP
     MOVEQ     #SMS.NTHG,D0   ; find next Thing
     TRAP      #DO.SMS2       ; if not SMS2, jump via HOTKEY System vector!!!
     MOVE.L    D0,-(SP)
     BSR       proc           ; process it
     MOVE.L    (SP)+,D0       ; was there another Thing?
     BNE.S     SDONE          ; no
     LEA       TH_NAME(A1),A0; point to this Thing's name
     BRA.S     SLOOP          ; and find the next Thing
SDONE
```

Trap #1 **D0=$2C** [SMS2] [EXT] SMS.NTHU

Next Thing User

Call parameters		Return parameters	
D1		D1	preserved
D2		D2	owner of usage block
D3		D3	preserved
		D4+	all preserved
A0	thing name	A0	preserved
A1	thing usage block or 0	A1	next usage block
A2		A2	smashed
		A3+	all preserved

Error returns:

ITNF Thing was not found
IJOB usage block was not found

This routine allows code to scan the usage list of a given Thing to find out which Jobs are using it. It returns in D2 the ID of the owner of the usage block passed. Note that the format of the usage block may change, so the returned address should only be used as a parameter for this routine. Note also that a Job may cease using the Thing between calls to this routine. The usage list of a Thing may be scanned thus:

```
     LEA       name,A0          ; point to Thing name
     SUB.L     A1,A1            ; start with first usage block
SLOOP
     MOVEQ     #SMS.NTHU,D0     ; find next user
     TRAP      #DO.SMS2         ; if not SMS2, jump via HOTKEY System vector!!!
     MOVE.L    D0,-(SP)
     BSR       proc             ; process this user
     MOVE.L    (SP)+,D0         ; was there another Thing?
     BEQ.S     SLOOP            ; yes!
SDONE
```

Trap #1 **D0=$33** [SMSQ] SMS.PSET

Set Printer Translate

For details on this trap call, refer to section 19, "Language dependent Modules".

SMS.RCHP

Trap #1 D0=$19

Release common heap area

Call parameters		Return parameters	
D1.L		D1	???
D2.L		D2	???
D3		D3	???
		D4+	all preserved
A0	base of area to be freed	A0	???
A1		A1	???
A2		A2	???
A3		A3	???
		A4+	all preserved

SMS.REHP

Trap #1 D0=$D

Link a free space (back) into a heap

Call parameters		Return parameters	
D1.L	length to link in	D1	???
D2		D2	???
D3		D3	???
		D4+	all preserved
A0	base of new space	A0	???
A1	ptr to ptr to free space	A1	???
A2		A2	???
A3		A3	???
A6	base address	A6	preserved

A6 is used as a base address for this call and for **SMS.ALHP** so that A0 (and A1) is an address relative to A6.

Trap #1 D0=$1B / D0=$1D / D0=$1F / D0=$21 / D0=$23 — SMS.REXI

D0=$1B	SMS.REXI
D0=$1D	**SMS.RPOL**
D0=$1F	**SMS.RSHD**
D0=$21	**SMS.RIOD**
D0=$23	**SMS.RFSD**

$1B Remove an external interrupt service routine
$1D Remove a polling 50/60 Hz service routine
$1F Remove a scheduler loop task
$21 Remove an IO device driver
$23 Remove a directory device driver from the operating system

Call parameters		Return parameters	
D1		D1	preserved
D2		D2	preserved
D3		D3	preserved
		D4+	all preserved
A0	address of link	A0	preserved
A1		A1	???
A2		A2	preserved
A3		A3	preserved

Trap #1 D0=$4 — SMS.RMJB

Remove job from transient program area

Call parameters		Return parameters	
D1.L	job ID	D1	???
D2		D2	???
D3.L	error code	D3	???
		D4+	all preserved
A0		A0	???
A1		A1	???
A2		A2	???
A3		A3	???
		A4+	all preserved

Error returns:

IJOB job does not exist
NC job not inactive

This trap removes a job (and its subsidiaries) from the transient program area. Only inactive jobs may be removed.

Trap #1	**D0=$17**		SMS.RMPA
	Release BASIC program area		
Call parameters		Return parameters	
D1.L	number of bytes to release	D1.L	nr. of bytes releaseed
D2		D2	???
D3		D3	???
		D4+	all preserved
A0		A0	???
A1		A1	???
A2		A2	???
A3		A3	???
A6	base address	A6	new base address
A7	user stack pointer	A7	new stack pointer

Trap #1	**D0=$13**		SMS.RRTC
	Read real-time-clock		
Call parameters		Return parameters	
D1		D1.L	time in seconds
D2		D2	???
D3		D3	preserved
		D4+	all preserved
A0		A0	???
A1		A1	preserved
A2		A2	preserved
		A3+	all preserved

The time returned in D1 is the number of seconds since 00:00 1 January 1961.

Trap #1 D0=$27 [SMS2] [EXT] SMS.RTHG

Remove Thing from list

Call parameters		Return parameters	
D1		D1	preserved
D2		D2	preserved
D3		D3	preserved
		D4+	all preserved
A0	name of Thing to remove	A0	preserved
A1		A1	preserved
A2		A2	preserved
		A3+	all preserved

Error returns:

FDIU Thing is in use
ITNF Thing not found

This routine removes a Thing from the system, if it is not in use. It will be of use where a different version of someThing is required. The Thing linkage block will have been returned to the common heap if this call succeeds.

Trap #1 D0=$38 [SMSQ] SMS.SCHP

Shrink allocation in common heap

Call parameters		Return parameters	
D1.L	new size required	D1.L	new size retained
D2		D2	???
D3		D3	???
		D4+	all preserved
A0	base address of area	A0	base address of area
A1		A1	???
A2		A2	???
A3		A3	???
		A4+	all preserved

Error returns (Z flag is not always set correctly):

IJOB job does not exist

This trap can be used to link part of a heap allocation back into the free space list. The first part of the area, starting from the base address, stays the same, and the following space which is not required anymore is released. This trap can be used to avoid unnecessary re-allocation and copying, in case too much memory is taken.

Trap #1 D0=$B — SMS.SPJB

Change job priority

Call parameters		Return parameters	
D1.L	job ID	D1.L	job ID
D2.B	priority (0 to 127)	D2	preserved
D3		D3	preserved
		D4+	all preserved
A0		A0	smashed
A1		A1	preserved
		A2+	preserved

Error returns:

IJOB job does not exist

This call is used to change the priority of a job. If D1 is a negative word it will change the priority of the current job. Setting the priority to 0 will cause inactivation. This call re-enters the scheduler and so a job setting its own priority to zero will be immediately inactivated.

Warning: Contrary to other QDOS documentation, A0 is smashed - it does not return the base of the job control area.

Trap #1 D0=$3A — [SMSQ] SMS.SEVT

Send Event to Job

Call parameters		Return parameters	
D1	destination job ID	D1.l	destination job ID
D2.b	event(s) to notify	D2.b	preserved
		D3+	all preserved
		A0+	all preserved

Error returns:

IJOB job does not exist

The events in D2 are sent the the destination job. If the job is waiting for one of these events, the job is released, otherwise the all the events are pended.

Trap #1 D0=$14 — SMS.SRTC

Set real-time-clock

Call parameters		Return parameters	
D1.L	time in seconds	D1.L	time in seconds
D2		D2	???
D3		D3	???
		D4+	all preserved
A0		A0	???
A1		A1	preserved
A2		A2	preserved
		A3+	all preserved

The value in D1 has to be the number of seconds since 00:00 1 January 1961 to set the new time and date.

Trap #1 D0=$8 — SMS.SSJB

Suspend a job

Call parameters		Return parameters	
D1.L	job ID	D1.L	job ID
D2		D2	preserved
D3.W	timeout period	D3	preserved
		D4+	all preserved
A0		A0	base of job ctrl area
A1	address of flag byte or 0	A1	preserved
A2		A2	preserved
A3		A3	preserved
		A4+	all preserved

Error returns:

IJOB job does not exist

A job may be suspended for an indefinite period, or until a given time has elapsed. The timeout period is up to ($7FFF times the frame time).

If the job ID is a negative word, then the current job is suspended. The flag byte is cleared when the job is released. If there is no flag byte, then A1 should be 0. If the timeout period is specified as -1, then the suspension is indefinite; no other negative value should be used. If the job is already suspended, the suspension will be reset. All jobs are rescheduled.

Trap #1 D0=$24 [not SMS2] SMS.TRNS

Set translation table and error messages

Call parameters		Return parameters	
D1	ptr to translation table, -1 or 0 (or 1)	D1	???
D2.L	ptr to message table, -1 or 0	D2	???
D3		D3	???
		D4+	all preserved
A0		A0	???
A1		A1	???
A2		A2	???
A3		A3	???
		A4+	all preserved

Error returns:

IPAR table has invalid format or is on odd address

This trap is supported from QDOS V1.10 onwards. If D1 or D2 are 0, then no translation is used and the standard error messages are used. -1 leaves the values as it has been defined previously. If D1=1 then a local translation table is used, depending on the language of the ROM (not in UK or US ROMs).

[SMSQ] If D2 is not zero and it points to a message table with language code $4AFB, this address is used for message group 0. The printer translate tables are then set according to the value in D1 (see sms.pset).

Trap #1 D0=$9 SMS.USJB

Release a job

Call parameters		Return parameters	
D1.L	job ID	D1.L	job ID
D2		D2	preserved
D3		D3	preserved
		D4+	all preserved
A0		A0	base of job ctrl area
A1		A1	preserved
A2		A2	preserved
A3		A3	preserved
		A4+	all preserved

Error returns:

IJOB job does not exist

After this call all jobs are rescheduled.

The activity of jobs can be controlled by activation or by modification of the priority levels. A job at priority level 0 is inactive, at any other priority level it is active.

Trap #1 D0=$28 [SMS2] [EXT] SMS.UTHG

Use Thing

Call parameters		Return parameters	
D1	Job ID	D1	Job ID
D2	parameter	D2	returned result
D3	timeout	D3	version
		D4+	all preserved
A0	name of Thing to use	A0	preserved
A1		A1	address of Thing or Extension (if Thing is an Extension Thing)
A2	parameter	A2	pointer to Thing linkage
		A3+	all preserved

Error returns:

ITNF Thing was not found
NIMP Extension not found
any returns from Thing's USE code

Request the use of a Thing for a given Job. Various extra parameters may be required for the Thing's USE code to determine whether the request can be granted - it is up to the provider of the Thing to document what these parameters are. Similarly, extra results may be returned. For an Extension Thing, D2 should be 0 or the required Extension ID.

Trap #1 D0=$3B [SMSQ] SMS.WEVT

Wait for Event

Call parameters		Return parameters	
D2.b	event(s) to wait for	D2.b	event(s) causing return
D3.w	timeout (-1 is forever)	D3.w	preserved
		D4+	all preserved
		A0+	all preserved

Error returns:

none

The job waits for one or more of the events in D2 or the timeout. The events returned in D2 are removed from the job's pending event vector (event accumulator).

Trap #1 D0=$25	[SMSQ] SMS.XTOP
External Operation	

The code which follows the TRAP #1 is executed as if it was part of a system call. When this TRAP #1 is encountered, the registers are changed to A6 pointing to the system variables, A5 pointing to the stack frame (which contains D7.l, previous A5, previous A6) and the code is executed in Supervisor mode. The routine must finish in an RTS, which brings it back to user mode on return. It continues with the next program line after the RTS.

Trap #1 D0=$2A		[SMS2] [EXT] SMS.ZTHG	
Zap Thing			
Call parameters		Return parameters	
D1		D1	preserved
D2		D2	preserved
D3		D3	preserved
		D4+	all preserved
A0	name of Thing to zap	A0	preserved
A1		A1	???
A2		A2	preserved
		A3+	all preserved
Error returns:			
ITNF	Thing was not found		

This routine removes a Thing and all Jobs using it. The call may not return, if the Job that called it was removed as a result of the zap. Because of this, it may not be called from supervisor mode under QDOS. The Thing linkage block is returned to the common heap by this call.

Trap 1 Keys - numerical order with page reference

14.0 I/O Management Traps

IOA.CLOS

Trap #2 D0=$2

Close a channel

Call parameters		Return parameters	
D1		D1+	all preserved
A0	channel ID	A0	???
A1		A1	???
A2		A2+	all preserved

Error returns:

ICHN channel not open

[SMSQ] IOA.CNAM

Trap #2 D0=$6

Fetch channel name

Call parameters		Return parameters	
D1		D1	preserved
D2.w	max length of string	D2	preserved
		D3+	all preserved
A0	channel ID	A0	preserved
A1	ptr to buffer	A1	device name (QDOS-string)
A2		A2	preserved
		A3+	all preserved

Error returns:

ICHN channel not open
IPAR buffer too small

IOA.DELF

Trap #2 D0=$4

Delete a file

Call parameters		Return parameters	
D1.L	job ID (as file open!!)	D1	???
D2		D2	preserved
D3		D3	???
		D4+	all preserved
A0	pointer to file name	A0	???
A1		A1	???
A2		A2	???
		A3+	all preserved

Error returns:

ICHN	not opened - too many channels open
IMEM	out of memory
FDNF	file or device not found
INAM	bad file or device name

A0 should point to a standard QDOS string containing the full name of the device and file.

IOA.FRMT

Trap #2 D0=$3

Format a sectored medium

Call parameters		Return parameters	
D1		D1.W	number of good sectors
D2		D2.W	total number of sectors
D3		D3	preserved
		D4+	all preserved
A0	ptr to medium name	A0	???
A1		A1	???
A2		A2+	all preserved

Error returns:

IMEM	out of memory
FDNF	drive not found
FDIU	drive in use
FMTF	format failed

The medium name is in the form of a character count (word) followed by the ASCII characters of the drive name, the drive number, underscore, then up to 10 characters for the medium name. For example,

```
dc.w  13
dc.b  'FLP1_November'
```

Trap #2 D0=$1

IOA.OPEN

Open a channel

Call parameters		Return parameters	
D1.L	job ID	D1.L	job ID
D2		D2	preserved
D3.L	open-key	D3	preserved
	0 old (exclusive) file or device		
	1 old (shared) file		
	2 new (exclusive) file		
	3 new (overwrite) file		
	4 open directory		
A0	pointer to file name	A0	channel ID
A1		A1	???
A2		A2	preserved
A3		A3	preserved
		A4+	all preserved

Error returns:

ICHN	not opened - too many channels open
IJOB	job does not exist
IMEM	out of memory
FDNF	file or device not found
FEX	file already exists
FDIU	drive in use
INAM	bad file or device name
IPAR	invalid open-key

If the job ID is passed as a negative word (for example -1) then the channel will be associated with the current job.

The file or device name should be a string of ASCII characters. This string is preceded by a character count (word), A0 should point to this word (on a word boundary).

The error return "**INAM**" indicates that the name of the device has been recognised but that the additional information is incorrect, for example **CON_512y240**.

The open-key is usually ignored for access to any non-shared device: in practice, this is anything other than a file store. If the error code is non-zero then no channel has been opened.

In order to open an input pipe, D3.L must hold the output pipe channel ID instead of an open key.

Note that New (overwrite) is not currently supported for Microdrive files.

Trap #2 D0=$5 [SMSQ] **IOA.SOWN**

Set new owner of open channel

Call parameters		Return parameters	
D1.l	new owner job-ID	D1	preserved
D2		D2	preserved
		D3+	all preserved
A0	channel ID	A0	preserved
		A1+	all preserved

Error returns:

ICHN channel not open
IJOB job does not exist

Trap 2 Keys - numerical order with page reference

ioa.open	$01	OPEN IOSS channel	3
ioa.clos	$02	CLOSe IOSS channel	1
ioa.frmt	$03	FoRMaT medium on device	2
ioa.delf	$04	DELete file from device	2
ioa.sown	$05	Set OWNer of channel [SMSQ]	4
ioa.cnam	$06	fetch Channel NAMe [SMSQ]	1

15.0 I/O Access Traps

Every I/O trap which is not supported by the system (e.g. IOF.XINF without level 2 device drivers) returns the error IPAR.

IOB.ELIN

Trap #3 D0=$4

Edit a line of characters (console driver only)

Call parameters		Return parameters	
D1	cursor/line length	D1	cursor/line length
D2.W	length of buffer	D2	preserved
D3.W	timeout	D3.L	preserved
		D4+	all preserved
A0	channel ID	A0	preserved
A1	pointer to end of line	A1	pointer to end of line
A2		A2	preserved
A3		A3	preserved
		A4+	all preserved

Error returns:

- NC not complete
- ICHN channel not open
- OVFL buffer overflow

Thisis similar to the fetch line trap, except that the pointer A1 is always to the end of the line, D1 contains the current cursor position in the msw and the length of the line in the lsw and the line (from the current cursor position) is written out to the console when the call is made. The line should not have a terminating character when the trap is made, but the terminating character will be included in the character count on return. Enter (ASCII 10), cursor up or cursor down are all acceptable terminating characters.

IOB.FBYT

Trap #3 D0=$1

Fetch a byte

Call parameters		Return parameters	
D1		D1.B	byte fetched
D2		D2	preserved
D3.W	timeout	D3.L	preserved
		D4+	all preserved
A0	channel ID	A0	preserved
A1		A1	???
A2		A2	preserved
A3		A3	preserved
		A4+	all preserved

Error returns:

NC not complete
ICHN channel not open
EOF end of file

IOB.FLIN
IOB.FMUL

Trap #3 D0=$2 or $3

D0=$2 IOB.FLIN fetch a line of characters terminated by ASCII <LF> ($A)
D0=$3 IOB.FMUL fetch a string of bytes

Call parameters		Return parameters	
D1		D1.W	number of bytes fetched
D2.W	length of buffer	D2	preserved
D3.W	timeout	D3.L	preserved
		D4+	all preserved
A0	channel ID	A0	preserved
A1	base of buffer	A1	updated pointer to buffer
A2		A2	preserved
A3		A3	preserved
		A4+	all preserved

Error returns:

NC not complete
ICHN channel not open
EOF end of file
OVFL buffer overflow

The character count of a fetch a line trap includes the linefeed character if found.

IOB.SBYT

Trap #3 D0=$5

Send a byte

Call parameters		Return parameters	
D1.B	byte to be sent	D1	???
D2		D2	preserved
D3.W	timeout	D3.L	preserved
		D4+	all preserved
A0	channel ID	A0	preserved
A1		A1	???
A2		A2	preserved
A3		A3	preserved
		A4+	all preserved

Error returns:

NC not complete
ICHN channel not open
DVFL drive full
ORNG off window/paper etc.

IOB.SMUL

Trap #3 D0=$7

Send a string of bytes

Call parameters		Return parameters	
D1		D1.W	number of bytes sent
D2.W	number of bytes to be sent	D2.W	preserved
D3.W	timeout	D3.L	preserved
		D4+	all preserved
A0	channel ID	A0	preserved
A1	base of buffer	A1	updated pointer to buffer
A2		A2	preserved
A3		A3	preserved
		A4+	all preserved

Error returns:

NC not complete
ICHN channel not open
DVFL drive full

Please refer to section 5.3.3 for details of the special treatment afforded to newlines on the console or screen device.

IOB.TEST

Trap #3 D0=$0

Check for pending input

Call parameters		Return parameters	
D1		D1	???
D2		D2	preserved
D3.W	timeout	D3.L	preserved
		D4+	all preserved
A0	channel ID	A0	preserved
A1		A1	???
A2		A2	preserved
A3		A3	preserved
		A4+	all preserved

Error returns:

NC not complete
ICHN channel not open
EOF end of file

This trap is used to check for pending input on a channel. It does not read any data or modify the input channel in any way. This only works on a console device if D3=0 and the keyboard queue is already connected to the console.

IOF.CHEK

Trap #3 D0=$40

Check all pending operations on a file

Call parameters		Return parameters	
D1		D1	???
D2		D2	preserved
D3.W	timeout	D3.L	preserved
		D4+	all preserved
A0	channel ID	A0	preserved
A1		A1	???
A2		A2	preserved
A3		A3	preserved
		A4+	all preserved

Error returns:

NC not complete
ICHN channel not open

This trap is used to check whether all of the pending operations have completed.

Trap #3 D0=$4C [EXT] [DD2] **IOF.DATE**

Set or read file date

Call parameters		Return parameters	
D1.l	Set/read key -1, 0 or date	D1.l	date set or read
D2.b	0 update date 2 backup date	D2	preserved
D3.w	timeout	D3	preserved
A0	channel ID	A0	preserved
A1		A1	preserved

Error returns:

Any I/O sub system errors

The update date of a file is usually set when a file which has been modified (including new copies of files) is closed (or flushed for the first time).

To read the appropriate date of a file, the trap should be called with the long word value -1 in d1. To set either the update date, or the backup date, of a file to the current date, the trap should be called with the value 0 in d1. A specific date may be set by calling the trap with required date in d1.
If the update date has been set by this trap, then the update date will not be re-set when the file is closed. The backup date is not stored in the file itself, and may be updated even if the file is open for read only.
The date is a long word giving the date and time in seconds from the start of 1961.
This trap is not supported on native QLs without Toolkit II, and it is partially supported on earlier floppy disc drivers. It should not be used on any other than Level 2 devices.

IOF.FLSH

Trap #3 D0=$41

Flush buffer for this file

Call parameters		Return parameters	
D1		D1	???
D2		D2	preserved
D3.W	timeout	D3.L	preserved
		D4+	all preserved
A0	channel ID	A0	preserved
A1		A1	???
A2		A2	preserved
A3		A3	preserved
		A4+	all preserved

Error returns:

NC not complete
ICHN channel not open

When a write operation to a file is complete, the data written may still be in the slave blocks rather than on the file. For further details please see Section 5.2 on File I/O. This call may be used to check that a file is in a known state.

IOF.LOAD

Trap #3 D0=$48

Load a file into memory

Call parameters		Return parameters	
D1		D1	???
D2.L	length of file	D2	preserved
D3.W	timeout	D3.L	preserved
		D4+	all preserved
A0	channel ID	A0	preserved
A1	base address for load	A1	top address after load
A2		A2	preserved
A3		A3	preserved
		A4+	all preserved

Error returns:

ICHN channel not open

Files may be loaded into memory in their entirety with the file load trap. If the transient program area is used for this, a trap #1 must have been invoked to reserve the space before the file load trap is invoked.

D3 should be set to -1 before this trap and the base address in A1 must be even.

IOF.MINF

Trap #3 D0=$45

Get information about medium

Call parameters		Return parameters	
D1		D1.L	empty/good sectors
D2		D2	preserved
D3.W	timeout	D3.L	preserved
		D4+	all preserved
A0	channel ID	A0	preserved
A1	ptr to 10 byte buffer	A1	end of medium name
A2		A2	preserved
A3		A3	preserved
		A4+	all preserved

Error returns:

NC not complete
ICHN channel not open

The name of the medium, its capacity, and the available space may be obtained for a file or directory that is open.

The medium name is 10 bytes long and left justified. Any remaining bytes are filled with the space character ($20).

The number of empty sectors is in the most significant word (msw) of D1, the total available on the medium is in the least significant word (lsw).

A sector is 512 bytes.

Trap #3 D0=$4D	[DD2] IOF.MKDR

Make directory

Call parameters		Return parameters	
D1.l	0	D1	preserved
D2		D2	preserved
D3.w	timeout should be -1	D3	preserved
A0	channel ID	A0	preserved
A1		A1	preserved

Error returns:

Any I/O sub system errors

The IOF.MKDR trap is called to convert the file into a directory.
The file itself should be empty. Any existing files which would, by virtue of their name, belong in the new directory, are transferred into the directory. The trap will return a 'bad parameter' error if the file is not empty.
The file must have been opened with a READ/WRITE access key (OLD, NEW or OVER); after this call the access mode of the file is changed to IOA.KDIR.

Trap #3 D0=$42	IOF.POSA

Position file pointer absolute

Call parameters		Return parameters	
D1.L	file position	D1.L	new file position
D2		D2	preserved
D3.W	timeout	D3.L	preserved
		D4+	all preserved
A0	channel ID	A0	preserved
A1		A1	???
A2		A2	preserved
A3		A3	preserved
		A4+	all preserved

Error returns:

NC not complete
ICHN channel not open
EOF end of file

If the position file pointer call is made for a direct sector access channel, a "special" file position flag can be specified in D1:

iofp.off $F0FFF0FF returns the sector offset of the first physical sector of the current partition on multiple-partition devices [SMSQ V2.77+], otherwise returns D1 unchanged

Trap #3 D0=$43

IOF.POSR

Position file pointer relative

Call parameters		Return parameters	
D1.L	offset to file pointer	D1.L	new file position
D2		D2	preserved
D3.W	timeout	D3.L	preserved
		D4+	all preserved
A0	channel ID	A0	preserved
A1		A1	???
A2		A2	preserved
A3		A3	preserved
		A4+	all preserved

Error returns:

NC not complete
ICHN channel not open
EOF end of file

If a file positioning trap returns an off file limits error, then the pointer is set to the nearest limit, this being 0 or end of file. The relative file positioning may, of course, be used to read the current file position.

IOF.RHDR

Trap #3 D0=$47

Read file header

Call parameters		Return parameters	
D1		D1.W	length of header read
D2.W	buffer length	D2	preserved
D3.W	timeout	D3.L	preserved
		D4+	all preserved
A0	channel ID	A0	preserved
A1	base of read buffer	A1	top of read buffer
A2		A2	preserved
A3		A3	preserved
		A4+	all preserved

Error returns:

NC not complete
ICHN channel not open
OVFL buffer overflow

The read header call is provided so that a job can allocate the space for a load call as well as determining the characteristics of a file. The buffer provided must be at least 14 bytes long, but should be minimum 16 for Level 2 drivers. In the case of a trap to a pure serial device, the length of the header returned in D1 will be spurious.

The file pointer is such that position zero is the first byte after the header. Thus block boundaries on standard directory driver files are at position 512*n-64.

Section 7 contains details about the format of a file header.

IOF.RNAM

Trap #3 D0=$4A [EXT] [DD2]

Rename file

Call parameters		Return parameters	
D1		D1	???
D2		D2	preserved
D3.w	timeout	D3	preserved
A0	channel ID	A0	preserved
A1	pointer to new filename (string)	A1	???

Error returns:

Any I/O sub system errors

This call renames a file. The name should include the drive name (e.g. FLP1_NEW_NAME).
This trap does not work on every device, especially not on MDV on an unexpanded QL.

IOF.SAVE

Trap #3 D0=$49

Save file from memory

Call parameters		Return parameters	
D1		D1	???
D2.L	length of file	D2	preserved
D3.W	timeout	D3.L	preserved
		D4+	all preserved
A0	channel ID	A0	preserved
A1	base address of file	A1	top address of file
A2		A2	preserved
A3		A3	preserved
		A4+	all preserved

Error returns:

ICHN channel not open
DRFL drive full

D3 should be set to -1 before this trap, and **IOF.LOAD**, and the base address in A1 must be even.

IOF.SHDR

Trap #3 D0=$46

Set file header

Call parameters		Return parameters	
D1		D1.W	length of header set
D2		D2	preserved
D3.W	timeout	D3.L	preserved
		D4+	all preserved
A0	channel ID	A0	preserved
A1	base of header def	A1	end of header def
A2		A2	preserved
A3		A3	preserved
		A4+	all preserved

Error returns:

NC not complete
ICHN channel not open

This call sets the first 14 bytes of the header. The length of file will normally be overwritten by the filing system. When a header is sent over a pure serial device, the 14 bytes of the header are preceded by a byte $FF.

Trap #3 D0=$4B [EXT] [DD2] IOF.TRNC

Truncate file

Call parameters		Return parameters	
D1		D1	???
D2		D2	preserved
D3.w	timeout	D3	preserved
A0	channel ID	A0	preserved
A1		A1	???

Error returns:

Any I/O sub system errors

This call truncates a file to the current byte position.
This trap does not work on every device, especially not on MDV on an unexpanded QL.

Trap #3 D0=$4E [DD2] IOF.VERS

Set or read file version

Call parameters		Return parameters	
D1.l	Set/read key -1, 0, version	D1.l	file version
D2		D2	preserved
D3.w	timeout	D3	preserved
A0	channel ID	A0	preserved
A1		A1	preserved

Error returns:

Any I/O sub system errors

To read the file version number, this trap should be called with the long word value -1 in d1. To preserve the file version number, this trap should be called with the value 0 in d1. To set a specific version number the trap should be called with the version number 1 to 65535 as a long word value in d1. If this trap is called to set the version number, the version number will not be incremented when the file is closed or flushed.
This trap is supported on Level 2 devices only.

Trap #3	D0=$4F		[DD2] IOF.XINF

Get extended information

Call parameters		Return parameters	
D1	0	D1	preserved
D2		D2	preserved
D3.w	timeout	D3	preserved
A0	channel ID	A0	preserved
A1	pointer to info buffer	A1	preserved

Error returns:

Any I/O sub system errors

This call fetches extended filing system information in a block 64 bytes long.

IOI_NAME	$00	string	up to 20 character medium name (null filled)
IOI_DNAM	$16	string	up to 4 character long device name (e.g. WIN)
IOI_DNUM	$1C	byte	drive number
IOI_RDON	$1D	byte	non zero if read only
IOI_ALLC	$1E	word	allocation unit size (in bytes)
IOI_TOTL	$20	long	total medium size (in allocation units)
IOI_FREE	$24	long	free space on medium (in allocation units)
IOI_HDRL	$28	long	file header length (per file storage overhead)
IOI_FTYP	$2C	byte	format type (1=QDOS, 2=MSDOS etc)
IOI_STYP	$2D	byte	format sub-type
IOI_DENS	$2E	byte	density
IOI_MTYP	$2F	byte	medium type (RAM=0, FLP=1, HD=2, CD=3)
IOI_REMV	$30	byte	set if removable
IOI_XXXX	$31	$0F bytes	set to -1

The number of allocation units required to store a file may be calculated as:
(file + header length + alloc unit size - 1) / (alloc unit size)

This trap is supported on Level 2 device drivers. It should be called to find out whether the current device is Level 2 or not and to check which operations are supported. If this trap succeeds, all other filing system traps will be available.

Trap #3	D0=$30	draw dot	IOG.DOT
	D0=$31	draw line	**IOG.LINE**
	D0=$32	draw arc	**IOG.ARC**
	D0=$33	draw ellipse	**IOG.ELIP**
	D0=$34	set graphics scale	**IOG.SCAL**
	D0=$36	set graphics cursor position	**IOG.SGCR**

Call parameters		Return parameters	
D1		D1	???
D2		D2.L	preserved
D3.W	timeout	D3.L	preserved
		D4+	all preserved
A0	channel ID	A0	preserved
A1	arithmetic stack pointer	A1	???
A2		A2	preserved
A3		A3	preserved
		A4+	all preserved

Error returns:

NC not complete
ICHN channel not open

Plot a point, line, arc, ellipse, set scale or graphics cursor position. Expects parameters on the arithmetic stack pointed to by (A1).

The first four traps (IOG.DOT, IOG.LINE, IOG.ARC and IOG.ELIP) draw various lines and arcs in the given window. Any point on these lines which fall outside the window will not be plotted.

All six traps expect parameters on the arithmetic stack pointed to by (A1). The format of the parameters required is as follows:

IOG.DOT	$00(A1)	y-coordinate
	$06(A1)	x-coordinate
IOG.LINE	$00(A1)	y-coord of finish of line
	$06(A1)	x-coord of finish of line
	$0C(A1)	y-coord of start of line
	$12(A1)	x-coord of start of line
IOG.ARC	$00(A1)	angle subtended by arc
	$06(A1)	y-coord of finish of line
	$0C(A1)	x-coord of finish of line
	$12(A1)	y-coord of start of line
	$18(A1)	x-coord of start of line
IOG.ELIP	$00(A1)	rotation angle
	$06(A1)	radius of ellipse
	$0C(A1)	eccentricity of ellipse
	$12(A1)	y-coord of centre
	$18(A1)	x-coord of centre

IOG.SCAL	$00(A1)	y position of bottom line of window
	$06(A1)	x position of left hand pixel of window
	$0C(A1)	length of Y axis (height of window)

IOG.SGCR	$00(A1)	graphics x-coordinate
	$06(A1)	graphics y-coordinate
	$0C(A1)	pixel offset to right
	$12(A1)	pixel offset down

For all the graphics traps, the parameters on the A1 stack are floating point, and the coordinates are specified in relation to an arbitrary origin (default is 0,0) with an arbitrary scale (default is: height of window = 100 units).

The calling program must allocate at least 240 bytes on the A1 stack.

IOG.FILL

Trap #3 D0=$35

Turn area flood on and off

Call parameters		Return parameters	
D1.L	key: 0=end flood 1=start or restart flood	D1	???
D2		D2.L	preserved
D3.W	timeout	D3.L	preserved
		D4+	all preserved
A0	channel ID	A0	preserved
A1		A1	???
A2		A2	preserved
A3		A3	preserved
		A4+	all preserved

Error returns:

NC not complete
ICHN channel not open

Trap #3 D0=$2E

IOW.BLOK

Fill rectangular block in window

Call parameters		Return parameters	
D1.B	colour	D1	???
D2		D2.L	preserved
D3.W	timeout	D3.L	preserved
		D4+	all preserved
A0	channel ID	A0	preserved
A1	base of block definition	A1	???
A2		A2	preserved
A3		A3	preserved
		A4+	all preserved

Error returns:

- NC not complete
- ICHN channel not open
- ORNG block falls outside window

This trap fills a rectangular block of a window with the current ink colour, taking into account the mode set by **IOW.SOVA**.

The block definition is in the same form as a window definition. It is 4 words long: width, height, X-origin and Y-origin. The origin is in relation to the window origin in which the block is to be drawn.

This is a fast way of drawing horizontal or vertical lines.

Trap #3 D0=$B

IOW.CHRQ

Return the current window size and cursor position in character coordinates

Call parameters		Return parameters	
D1		D1	???
D2		D2.L	preserved
D3.W	timeout	D3.L	preserved
		D4+	all preserved
A0	channel ID	A0	preserved
A1	base of enquiry block	A1	???
A2		A2	preserved
A3		A3	preserved
		A4+	all preserved

Error returns:

NC not complete
ICHN channel not open

The window size (X,Y) and cursor position (X,Y) are put into a 4 word enquiry block. The top left hand corner of the window is cursor position 0,0. This trap activates the newline if pending in the window.

Trap #3	D0=$20	clear all of window	IOW.CLRA
	D0=$21	clear top of window	**IOW.CLRT**
	D0=$22	clear bottom of window	**IOW.CLRB**
	D0=$23	clear cursor line	**IOW.CLRL**
	D0=$24	clear right hand end of cursor line	**IOW.CLRR**

Call parameters		Return parameters	
D1		D1	???
D2		D2.L	preserved
D3.W	timeout	D3.L	preserved
		D4+	all preserved
A0	channel ID	A0	preserved
A1		A1	???
A2		A2	preserved
A3		A3	preserved
		A4+	all preserved

Error returns:

NC not complete
ICHN channel not open

The clear window traps can clear all or part of a window. To clear a part of a window the cursor is used as a reference. The clear operation consists of overwriting all the pixels in the designated area with paper colour.

The division between the top of the window and the bottom of the window is the cursor line. The cursor line is neither the top nor the bottom of the window.

The cursor line is the whole height of the current character fount (either 10 or 20 rows). The right hand end includes the character at the current cursor position.

Trap #3 D0=$F

IOW.DCUR

Disable the cursor

Call parameters		Return parameters	
D1		D1	???
D2		D2.L	preserved
D3.W	timeout	D3.L	preserved
		D4+	all preserved
A0	channel ID	A0	preserved
A1		A1	???
A2		A2	preserved
A3		A3	preserved
		A4+	all preserved

Error returns:

NC not complete
ICHN channel not open

The call to suppress the cursor does not return an error if the cursor is already suppressed, as it merely ensures that the cursor is in the desired state.

Trap #3 D0=$C	IOW.DEFB

Set the border width and colour

Call parameters		Return parameters	
D1.B	colour	D1	???
D2.W	width	D2.L	preserved
D3.W	timeout	D3.L	preserved
		D4+	all preserved
A0	channel ID	A0	preserved
A1		A1	???
A2		A2	preserved
A3		A3	preserved
		A4+	all preserved

Error returns:

NC not complete
ICHN channel not open

This call redefines the border of a window. By default this is of no width. The width of the border is doubled on the vertical edges. The border is inside the window limits. All subsequent screen traps (except this one) use the reduced window size for defining cursor position and window limits.

As a special case, the colour $80 defines a transparent border so that the border contents are not altered by the trap.

If the call changes the width of the border, then the cursor is reset to the home position (top left hand corner).

IOW.DEFW

Trap #3 D0=$D

Redefine a window

Call parameters		Return parameters	
D1.B	border colour	D1	???
D2.W	border width	D2.L	preserved
D3.W	timeout	D3.L	preserved
		D4+	all preserved
A0	channel ID	A0	preserved
A1	base of window block	A1	???
A2		A2	preserved
A3		A3	preserved
		A4+	all preserved

Error returns:

NC not complete
ICHN channel not open
ORNG window does not fit on page

This call redefines the shape or position of a window: the contents are not moved or modified, but the cursor is repositioned at the top left hand corner of the new window. The window block is 4 words long representing the width, height, X origin and Y origin.

IOW.DONL

Trap #3 D0=$2F

Do a pending newline

Call parameters		Return parameters	
D1		D1	???
D2		D2.L	preserved
D3.W	timeout	D3.L	preserved
		D4+	all preserved
A0	channel ID	A0	preserved
A1		A1	???
		A2+	all preserved

Error returns:

NC not complete
ICHN channel not open

This trap forces a newline pending in a window to be carried out. This is normally where something has been printed at the bottom of a window, but the newline has not been performed as this would cause the window to scroll upwards. If a newline is not pending in the window, then the routine will return without affecting the display, otherwise the screen is scrolled upwards SD_YINC pixels (if necessary) and the cursor is placed at the start of the next line.

Trap #3 D0=$E

IOW.ECUR

Enable the cursor

Call parameters		Return parameters	
D1		D1	???
D2		D2.L	preserved
D3.W	timeout	D3.L	preserved
		D4+	all preserved
A0	channel ID	A0	preserved
A1		A1	???
A2		A2	preserved
A3		A3	preserved
		A4+	all preserved

Error returns:

NC not complete
ICHN channel not open

The call to enable the cursor does not return an error if the cursor is already enabled, as it merely ensures that the cursor is in the desired state.

IOW.FONT

Trap #3 D0=$25

Set or reset the fount

Call parameters		Return parameters	
D1		D1	???
D2	0 or "DEFF"		D2.L preserved
D3.W	timeout	D3.L	preserved
		D4+	all preserved
A0	channel ID	A0	preserved
A1	base of fount	A1	???
A2	base of second fount	A2	preserved
A3		A3	preserved
		A4+	all preserved

Error returns:

NC not complete
ICHN channel not open

The fount is a 5x9 array of pixels in a 6x10 rectangle. A default fount and a second fount are built into the ROM, although alternative founts may be selected.

If either fount address is given as zero, the relevant default fount will be used.

The structure of a fount assumes that up to a certain value characters are invalid (default $1E), from the next value (default $1F) a known number of characters are valid (default $61). Thus the structure is as follows:

$00	lowest valid character (byte)
$01	number of valid characters-1 (byte)
$02 to $0A	9 bytes of pixels for the first valid character
$0B to $13	etc.

Each byte of pixels has the pixels in bit 6 to 2 (inclusive) of the byte. The top row of any character is implicitly blank.

If a character, which is to be written, is found to be invalid in the first fount, it is written using the second fount. If it is also invalid in the second fount, then the lowest valid character of the second fount is used.

The default fount extends from $20 to $7F.

In SMSQ, an optional parameter can be specified in D2. If it contains the ASCII string "DEFF", then this call sets the default system fount used by any subsequently opened channels.

Trap #3	D0=$1B pan all of window	IOW.PANA
	D0=$1E pan cursor line	**IOW.PANL**
	D0=$1F pan right hand end of cursor line	**IOW.PANR**

Call parameters		Return parameters	
D1.W	distance to pan	D1	???
D2		D2.L	preserved
D3.W	timeout	D3.L	preserved
		D4+	all preserved
A0	channel ID	A0	preserved
A1		A1	???
A2		A2+	preserved

Error returns:

NC not complete
ICHN channel not open

The whole of a window, or the whole of the cursor line, or the right hand end of the cursor line may be panned by any number of pixels to the right or to the left. A positive distance implies that the pixels will move to the right. The space left behind will be filled with paper colour.

The cursor line is the whole height of the current character fount (either 10 or 20 rows). The right hand end includes the character at the current cursor position.

Trap #3	D0=$A	IOW.PIXQ

Return the current window size and cursor position in pixel coordinates

Call parameters		Return parameters	
D1		D1	???
D2		D2.L	preserved
D3.W	timeout	D3.L	preserved
		D4+	all preserved
A0	channel ID	A0	preserved
A1	base of enquiry block	A1	???
A2		A2	preserved
A3		A3	preserved
		A4+	all preserved

Error returns:

NC not complete
ICHN channel not open

The window size (X,Y) and cursor position (X,Y) are put into a 4 word enquiry block. The top left hand corner of the window is cursor position 0,0. This trap activates the newline if pending in the window.

IOW.RCLR

Trap #3 D0=$26

Recolour a window

Call parameters		Return parameters	
D1		D1	???
D2		D2.L	preserved
D3.W	timeout	D3.L	preserved
		D4+	all preserved
A0	channel ID	A0	preserved
A1	pointer to colour list	A1	???
A2		A2	preserved
A3		A3	preserved
		A4+	all preserved

Error returns:

NC not complete
ICHN channel not open

A window may be recoloured without changing the information in it. This allows the same sort of effects as resetting the attributes of an attribute based screen, but it is very much slower.

The colour list is 8 bytes long and should contain the new colours required for each of the 8 colours in the window. Each of the new colours must be in the range 0 to 7. For 4 colour mode, only bytes 0, 2, 4 and 6 need to be filled in.

Trap #3	D0=$18	scroll all of window	IOW.SCRA
	D0=$19	scroll top of window	**IOW.SCRT**
	D0=$1A	scroll bottom of window	**IOW.SCRB**

Call parameters		Return parameters	
D1.W	distance to scroll	D1	???
D2		D2.L	preserved
D3.W	timeout	D3.L	preserved
		D4+	all preserved
A0	channel ID	A0	preserved
A1		A1	???
A2		A2	preserved
A3		A3	preserved
		A4+	all preserved

Error returns:

NC not complete
ICHN channel not open

Part or all of window may be scrolled; for partial scrolling the cursor is used as a reference. These traps cause pixels to be transferred from one row to another. Vacated rows of pixels are filled with paper colour. A positive scroll distance implies that the pixels in the window will be moved in a positive direction, i.e. downwards. The space left behind will be filled with paper colour.

The division between the top of the window and the bottom of the window is the cursor line. The cursor line is included in neither the top nor the bottom of the window. The cursor is not moved.

Trap #3	D0=$10	set cursor position by character intervals	IOW.SCUR
	D0=$11	set cursor column	**IOW.SCOL**
	D0=$12	put cursor on a new line	**IOW.NEWL**
	D0=$13	move cursor to previous column	**IOW.PCOL**
	D0=$14	move cursor to next column	**IOW.NCOL**
	D0=$15	move cursor to previous row	**IOW.PROW**
	D0=$16	move cursor to next row	**IOW.NROW**

Call parameters		Return parameters	
D1.W	column number (D0=10,11)	D1	???
D2.W	row number (D0=10)	D2.L	preserved
D3.W	timeout	D3.L	preserved
		D4+	all preserved
A0	channel ID	A0	preserved
A1		A1	???
A2		A2	preserved
A3		A3	preserved
		A4+	all preserved

Error returns:

NC not complete
ICHN channel not open
ORNG position would be out of window

In the case of an error return, the cursor position is not changed. The cursor position is the top left hand corner of the next character rectangle in relation to the top left hand corner of the window. These traps clear the pending newline in the window.

IOW.SFLA
IOW.SULA

Trap #3 D0=$2A set flash attribute
D0=$2B set underline attribute

Call parameters		Return parameters	
D1.B	0=attribute off, else attribute on	D1	???
D2		D2.L	preserved
D3.W	timeout	D3.L	preserved
		D4+	all preserved
A0	channel ID	A0	preserved
A1		A1	???
A2		A2	preserved
A3		A3	preserved
		A4+	all preserved

Error returns:

NC not complete
ICHN channel not open

IOW.SOVA

Trap #3 D0=$2C set the character writing or plotting mode

Call parameters		Return parameters	
D1.W	mode: -1 ink is exclusive ored into the background 0 character background is strip colour 1 character background is transparent	D1	???
D2		D2.L	preserved
D3.W	timeout	D3.L	preserved
		D4+	all preserved
A0	channel ID	A0	preserved
A1		A1	???
A2		A2	preserved
A3		A3	preserved
		A4+	all preserved

Error returns:

NC not complete
ICHN channel not open

Mode 0 or 1 plotting is in ink colour.

Trap #3	D0=$27	set paper colour	IOW.SPAP
	D0=$28	set strip colour	**IOW.SSTR**
	D0=$29	set ink colour	**IOW.SINK**

Call parameters		Return parameters	
D1.B	colour	D1	???
D2		D2.L	preserved
D3.W	timeout	D3.L	preserved
		D4+	all preserved
A0	channel ID	A0	preserved
A1		A1	???
A2		A2	preserved
A3		A3	preserved
		A4+	all preserved

Error returns:

NC not complete
ICHN channel not open

The screen driver uses three colours. There is the background colour of a window: referred to as paper colour; this is the colour which is used by the scroll, pan and clear operations. There is the colour which is used by the character generator to provide a highlighting background for individual characters or words: referred to as strip colour. Finally, there is the colour used for writing characters and drawing graphics: referred to as ink colour.

Trap #3	D0=$17	set cursor to pixel position	IOW.SPAP

Call parameters		Return parameters	
D1.W	x-coordinate	D1	???
D2.W	y-coordinate	D2.L	preserved
D3.W	timeout	D3.L	preserved
		D4+	all preserved
A0	channel ID	A0	preserved
A1		A1	???
A2		A2	preserved
		A4+	all preserved

Error returns:

NC not complete
ICHN channel not open
ORNG off window

The cursor position is the top left hand corner of the next character rectangle referred to the top left hand corner of the window. This trap clears the pending newline in the window.

Trap #3 D0=$2D set character size and spacing — IOW.SSIZ

Call parameters		Return parameters	
D1.W	character width/spacing 0 single width, 6 pixel spacing 1 single width, 8 pixel spacing 2 double width, 12 pixel spacing 3 double width, 16 pixel spacing	D1	???
D2.W	character heigth/spacing 0 single height, 10 pixel spacing 1 double height, 20 pixel spacing	D2.L	preserved
D3.W	timeout	D3.L	preserved
		D4+	all preserved
A0	channel ID	A0	preserved
A1		A1	???
A2		A2	preserved
A3		A3	preserved
		A4+	all preserved

Error returns:

NC not complete
ICHN channel not open

The character generator supports two widths and two heights of character. In 8 colour mode, only the double width characters may be used. In addition the spacing between characters is entirely flexible, but for simplicity of use only two additional spacings are supported directly: these are 8 pixel and 16 pixel, in single and double width respectively.

Calls with D1=0 or 1 in 8 colour mode will operate as though a call had been made with D1 equal to 2 or 3 respectively.

IOW.XTOP

Trap #3 D0=$9

Call an extended operation

Call parameters		Return parameters	
D1	parameter	D1	parameter
D2	parameter	D2.L	preserved
D3.W	timeout	D3.L	preserved
		D4+	all preserved
A0	channel ID	A0	preserved
A1	parameter	A1	parameter
A2	start address of routine	A2	preserved
A3		A3	preserved
		A4+	all preserved

Error returns:

NC not complete
ICHN channel not open
plus anything from the operation routine

This trap invokes an externally supplied routine as if it were part of the standard screen driver. D1, D2 and A1 are passed to the routine, while only D1 and A1 are returned. The code within the routine is executed in supervisor mode with A0 pointing to the channel definition block (see Section 7.2, 18.7 to 18.10) and A6 pointing to the system variables as for standard device drivers. Both A0 and A6 must not be smashed.

Trap 3 Keys - numerical order with page reference

iof.chek	$40	CHEcK all pending operations on file	4
iof.flsh	$41	FLuSH all buffers	6
iof.posa	$42	set file POSition to Absolute address	8
iof.posr	$43	move file POSition Relative to current position	9
iof.minf	$45	get Medium INFormation	7
iof.shdr	$46	Set file HeaDeR	11
iof.rhdr	$47	Read file HeaDeR	10
iof.load	$48	(scatter) LOAD file	6
iof.save	$49	(scatter) SAVE file	11
iof.rnam	$4a	ReNAMe file [EXT, DD2]	10
iof.trnc	$4b	TRuNCate file to current position [EXT, DD2]	12
iof.date	$4c	set or get file DATEs [EXT,DD2]	5
iof.mkdr	$4d	MaKe DiRectory [DD2]	8
iof.vers	$4e	set or get VERSion (d1 keys as iof.date) [DD2]	12
iof.xinf	$4f	get eXtended INFormation [DD2]	13

16.0 Vectored Routines

Vector $D6		[SMS]	CV.DATIL
Convert date and time to Integer Long			
Call parameters		Return parameters	
D1		D1	date
D2		D2	preserved
D3		D3	preserved
A0		A0	preserved
A1	ptr to 6 words	A1	???
A2		A2	preserved
A3		A3	preserved

This routine converts the single parameters year, month, day, hour, minute and second into the internal longword format.

This routine is not available on a standard QL or QL-Emulator.

Vector $100	Convert Decimal to Floating Point	CV.DECFP
$102	Convert Decimal to Integer (word)	**CV.DECIW**
$104	Convert Binary to Integer (byte) *	**CV.BINIB**
$106	Convert Binary to Integer (word) *	**CV.BINIW**
$108	Convert Binary to Integer (long) *	**CV.BINIL**
$10A	Convert Hexadecimal to Integer (byte) *	**CV.HEXIB**
$10C	Convert Hexadecimal to Integer (word) *	**CV.HEXIW**
$10E	Convert Hexadecimal to Integer (long) *	**CV.HEXIL**

Call parameters		Return parameters	
D1		D1	???
D2		D2	???
D3		D3	???
D7	0 or ptr to end of buffer	D7	preserved
A0	ptr to buffer (rel. A6)	A0	updated to end of buffer+1
A1	ptr to RI stack (rel. A6)	A1	updated
A2		A2	???
A3		A3	???

Error returns:

XP error in conversion (e.g. 1..0 as floating point or no digits or too many hex or binary digits)

All addresses passed to this routine must be relative to A6.

Utilities marked with * are non-functioning in QDOS V1.03 and earlier.

These routines convert from ASCII characters in a buffer to a value on the stack. Conversion ends either at the character to which D7 points (if given) or at an invalid character within the buffer.

The hex and binary conversions from ASCII to number, always put a long word on the A1 stack. A1 is set to point to the least significant byte or less significant word for the byte and word conversions.

The decimal conversions may use up to about 30 bytes on the A1 stack.

If there is an error then A0 and A1 are both unchanged.

Vector $F0	Convert Floating Point to Decimal	CV.FPDEC
$F2	Convert Integer (word) to Decimal	**CV.IWDEC**
$F4	Convert Integer (byte) to Binary	**CV.IBBIN**
$F6	Convert Integer (word) to Binary	**CV.IWBIN**
$F8	Convert Integer (long) to Binary	**CV.ILBIN**
$FA	Convert Integer (byte) to Hexadecimal	**CV.IBHEX**
$FC	Convert Integer (word) to Hexadecimal	**CV.IWHEX**
$FE	Convert Integer (long) to Hexadecimal	**CV.ILHEX**

Call parameters		Return parameters	
D1		D1	???
D2		D2	???
D3		D3	???
A0	ptr to buffer (rel. A6)	A0	ptr to buffer (rel. A6)
A1	ptr to RI stack (rel. A6)	A1	updated
A2		A2	???
A3		A3	???

All addresses passed to these routines must be relative to A6.

These routines convert a value on the stack to a set of ASCII characters in a buffer. For CV.FPDEC and CV.IWDEC, D1 contains the length of the result.

Vector $EC	get date and time	CV.ILDAT
$EE	get day of week	**CV.ILDAY**

Call parameters		Return parameters	
D1.L	date (interval value)	D1	preserved
D2		D2	preserved
D3		D3	preserved
A0		A0	preserved
A1	ptr to RI stack (rel. A6)	A1	updated
A2		A2	preserved
A3		A3	preserved

All addresses passed to this routine must be relative to A6.

There are two date conversion routines:
CV.ILDAT returns the date in the form
yyyy mmm dd hh:mm:ss

CV.ILDAY returns a three letter day of the week. The result is put on the A1 stack in string format. At least 22 bytes are required by **CV.ILDAT** and at least 6 bytes by **CV.ILDAY**.

IOQ.SETQ

Vector $DC	set up a queue	IOQ.SETQ
$DE	test status of queue	**IOQ.TEST**
$E0	put byte into queue	**IOQ.PBYT**
$E2	extract byte from queue	**IOQ.GBYT**
$E4	put end of file marker into queue	**IOQ.SEOF**

Call parameters		Return parameters	
D1.L	queue length or data	D1	data
D2		D2	preserved/free space
D3		D3	preserved
A0		A0	preserved
A1		A1	preserved
A2	pointer to queue	A2	preserved
A3		A3	???

Error returns:

NC queue is full (PBYT) or empty (GBYT, TEST)
EOF end of file reached (GBYT, TEST)

The data length should be less than 32767. A queue definition is given in section 18.10.

IOU.DNAM

Vector $122

Decode Device Name

Call parameters		Return parameters	
D1		D1	???
D2		D2	???
D3		D3	???
A0	pointer to name	A0	preserved
A1		A1	???
A2		A2	???
A3	pointer to parameters	A3	preserved

Error returns:

ITNF not recognised
INAM name recognised but bad parameters

This routine parses a device name. Given a device name and a description of the syntax of the name to be checked against and for the possible parameters to be appended to it, the routine determines whether the name is recognised, and extracts the parameters if it is. The device name is formed using four components:

Name	ASCII characters, normally letters. Case is ignored.
Separator	Single ASCII character. Case is ignored.
Number	Decimal number in the range 0 to 32767.
Code	One of a list of ASCII characters.

On entry to the routine, A0 must point to the device name to be checked (which is in the usual Qdos string format), A3 must point to an area of memory which is sufficient to hold the decoded parameter values, and A6 must point to the base of system variables. The device description starts 6 bytes after the call, and is in the following format:

word	number of characters in the device name to be checked for
bytes	the characters of the device name to be checked for (word-aligned)
word	number of parameters

The byte which then follow are the various parameters to be checked for. For each parameter to be checked, you will need to use one of the following options:

byte space, byte separator, word default value (numeric with separator)
word negative number, word default value (numeric with no separator)
word positive number of possible codes, bytes for the ASCII codes

Note that all letters must be in upper case.

For each numeric parameter value in the description, the utility will return either the value given in the device name, or the default. For each list of codes in the description the utility will return the position of the code in the list, or zero. All returned parameters are word length integers.

Examples:

The CON description is:

DC.W3,'CON'	console
DC.W5	five parameters
DC.W' _',448,' X',200	window size
DC.W' A',32,' X',16	window position
DC.W' _',128	keyboard queue length

Device name	Parameters
CON	448,200,0,0,128
CON_256	**256**,200,0,0,128
con__60	448,200,0,0,**60**
cona0x12	448,200,**0,12**,128
con_256x64a64x128_20	**256,64,64,128,20**

The SER description is:

DC.W3,'SER'	RS232 serial device
DC.W4	four parameters
DC.W-1,1	port number (default 1)
DC.W4,'OEMS'	parity (odd/even/mark/space)
DC.W2,'IH'	ignore/use handshaking
DC.W3,'RZC'	Raw/use CTRLZ/use CR

Device name	Parameters
SER	1,0,0,0
SERE	1,2,0,0
ser2miZ	2,3,1,2

If the name is not matched, the routine returns immediately after the call with **ERR.ITNF** in D0. If the name is matched but the additional information is incorrect, it returns 2 bytes after the call with **ERR.INAM** in D0. If a match is found, it returns 4 bytes after the call with D0=0 (on SMS and the Emulator), otherwise D0 is smashed.

Vector $E8 direct queue handling	**IOU.SSQ**
$EA general IO handling	**IOU.SSIO**

Call parameters		Return parameters	
D1	standard IOSS value	D1	standard IOSS value
D2	standard IOSS value	D2	standard IOSS value
D3	standard IOSS value	D3	???
A0	standard IOSS value	A0	preserved
A1	standard IOSS value	A1	standard IOSS value
A2		A2	???
A3		A3	???

Error returns:

IPAR undefined action
or errors returned from supplied routines

These routines must be called from supervisor mode, with A6 pointing to the base of system variables. It may not be called from a task which services an interrupt.

IOU.SSQ is a direct queue handling routine. When the channel definition block is set up for simple I/O then the 7th and 8th long words should be set to point to the queues for input and output respectively. If either input or output is prohibited, then the corresponding pointer should be zero.

IOU.SSIO should be called with the standard IOSS values in D0, D1, D2, D3, A0 and A1.

For serial I/O where the operations for byte input and output are not so simple, the routine **IOU.SSIO** may be called. The call instruction should be followed by three long words, these being the entry addresses for

testing for pending byte input, (next byte in D1)
fetch byte, (byte in D1)
send byte. (byte in D1)

The use of absolute addresses for these may prove awkward; so the entry to this routine is best included in the physical definition block for the driver:

```
at $28(A3) or similar                or

387800E8  MOVE.W   $E8,A4                          DC.L     TEST
4E94      JSR      (A4)                            DC.L     FETCH
          DC.L     TEST                            DC.L     SEND
          DC.L     FETCH                 4E75      RTS
          DC.L     SEND
4E75      RTS

invoked by                           or

          JSR      $28(A3)                         PEA      $28(A3)
                                                   MOVE.W   $E8,A4
                                                   JMP      (A4)
```

For the calls to the three service routines D0 should be returned as the error code, D1 to D3 and A1 to A3 inclusive are volatile.

Both of these calls treat actions 0, 1, 2, 3, 5 and 7, the header set and read actions and load and save; for undefined actions they return **ERR.IPAR**.

Vector $124	read a sector	[QL] **MD.READ**
$126	write a sector	[QL] **MD.WRITE**
$128	verify a sector	[QL] **MD.VERIF**
$12A	read a sector header	[QL] **MD.RDHDR**

Call parameters		Return parameters	
D1		D1	file nr (read/verify)
D2		D2	block nr (read/verify)
D7		D7	sector nr (read header)
A0		A0	???
A1	pointer to start of buffer	A1	pointer to end of buffer
A2		A2	???
A3	$18020	A3	$18020

Error returns:

MD.WRITE	none	
MD.READ, MD.VERIF	normal	failed
	return+2	OK
MD.RDHDR	normal	bad medium
	return+2	bad sector header
	return+4	OK

The microdrive support routines are vectored to simplify the writing of file recovery programs. On entry A3 must point to the microdrive control register, and the interrupts must be disabled. All registers except A3 and A6 are treated as volatile.

These routines do not set D0 on return but have multiple returns.

Before calling **MD.WRITE** the stack pointer must point to a word: the file number and the block number of the sector to be written are in the high and low byte respectively.

These vectors point to $4000 before the actual entry point. The following code may be used to read a header:

```
MOVE.W   D2,-(sp)          ; store block number and sector number on stack
MOVE.W   MD.RDHDR,An       ; Vector
JSR      $4000(An)
BRA.S    bad_medium        ; bad medium error handler
BRA.S    bad_sector        ; bad sector header handler
MOVEQ    #0,D0             ; all is fine
RTS
```

MEM.ACHP

Vector $C0

Allocate common heap area

Call parameters		Return parameters	
D1.L	space required	D1.L	space allocated
D2		D2	???
D3		D3	???
A0		A0	base of area allocated
A1		A1	???
A2		A2	???
A3		A3	???
A6	ptr to system variables	A6	???

Error returns:

IMEM out of memory
The condition code is not cleared on success on all ROM versions

This routine must be called from supervisor mode. It may not be called from a task which services an interrupt.

The space requested must include room for the heap entry header. For simple heap entries, this is 16 bytes long, for IOSS channels this is 24 bytes long.

The address of the heap area is the base of the area allocated, not the base of the area which may be used (contrast with TRAP #1, D0=$18 and $19).

The area allocated is cleared to zero.

MEM.ALHP

Vector $D8

Allocate an area in a heap

Call parameters		Return parameters	
D1.L	length required	D1.L	length allocated
D2		D2	???
D3		D3	???
A0	ptr to ptr to free space	A0	base of area allocated
A1		A1	???
A2		A2	???
A3		A3	???

Error returns:

IMEM no free space large enough

See section 4.1 for details of the heap allocation mechanism.

Vector $D2 link an item into a list $D4 unlink an item from a list			**MEM.LLST** **MEM.RLST**
Call parameters		Return parameters	
D1		D1	preserved
D2		D2	preserved
D3		D3	preserved
A0	base of item (un)linked	A0	preserved
A1	pointer to previous item	A1	updated
A2		A2	preserved
A3		A3	preserved

These routines are provided for handling linked lists.

These routines use A0 to pass the base address of the item to be linked or unlinked, and A1 to pass a pointer which points to either the pointer to the first item in the list, or to an item in the list.

When an item is linked in, it will be linked in at the start of the list, or, if A1 pointed to an item in the list, after that item. When starting a new list, A1 must be zero.

When an item is removed, A1 may point to the pointer to the first item in the list, or to any item in the list before the item to be removed.

When starting a new list, the pointer to the first item in the list must be zero.

Each item in the list must have 4 bytes reserved at the start for the link pointer.

An example of MEM.RLST is given in Section 7.2.2

Vector $C2			**MEM.RCHP**
	Release common heap space		
Call parameters		Return parameters	
D1		D1	???
D2		D2	???
D3		D3	???
A0	base of area to release	A0	???
A1		A1	???
A2		A2	???
A3		A3	???
A6	ptr to system variables	A6	???

This routine must be called from supervisor mode. It may not be called from a task which services an interrupt. See entry for **MEM.ACHP**.

MEM.REHP

Vector $DA

Link a free space (back) into a heap

Call parameters		Return parameters	
D1.L	length to link in	D1.L	???
D2		D2	???
D3		D3	???
A0	base of new space	A0	???
A1	ptr to ptr to free space	A1	???
A2		A2	???
A3		A3	???

OPW.WIND

Vector $C4 set up a window using a supplied name **OPW.WIND**
$C6 set up console window **OPW.CON**
$C8 set up screen window **OPW.SCR**

Call parameters		Return parameters	
D1		D1	???
D2		D2	???
D3		D3	???
A0	ptr to name (OPW.WIND only)	A0	channel ID
A1	ptr to parameter block	A1	???
A2		A2	???
A3		A3	???

Error returns:

INAM bad device name (WINDW only)
IMEM out of memory
ICHN out of channels
ORNG window is off-screen

The above three routines, which must be called in user mode, set up console or screen windows using a parameter list, pointed to by A1. In the first case, the window is opened using a name which has been supplied, a block of parameters defining the border, and the paper, strip and ink colours. The window is set up and cleared for use.

The parameter block is as follows:

$00	border colour (byte)
$01	border width (byte)
$02	paper/strip colour (byte)
$03	ink colour (byte)

For the second and third routines a further four words will need to be added to the parameter block to define the window:

$04	width (word)
$06	height (word)
$08	X-origin (word)
$0A	Y-origin (word)

Vector $11C executes an operation — **QA.OP**

$11E executes a list of operations — **QA.MOP**

Call parameters		Return parameters	
D0.W	operation (QA.OP)	D0.L	error code
D1		D1	preserved
D2		D2	preserved
D3		D3	preserved
A0		A0	preserved
A1	ptr to RI stack (rel. A6)	A1	updated
A2		A2	preserved
A3	absolute ptr to operation list (QA.MOP)	A3	preserved
A4	ptr to base of var area (rel. A6)	A4	preserved

Error returns:

OVFL arithmetic overflow

All addresses except A3 (for QA.MOP only) passed to these routines must be relative to A6.

The arithmetic package is available for general use through two vectors: the first executes a single operation, the second executes a list of operations.

The package operates on floating point numbers on a downward stack pointed to by (A6,A1.L). It operates on the top of the stack (TOS) which is pointed to by (A6,A1.L), and the next on the stack (NOS) at 6(A6,A1.L).

See section 9.5 for details of the floating point format.

There are two types of operation codes which can be passed to the interpreter to be executed.

Operation codes between $02 and $30 (inclusive) carry out various arithmetic operations on the stack, with the result being stored at 0(A6,A1.L).

Operation codes between $FFFF and $FF31 allow you to access intermediate results and variables stored on a second stack, the top of which is pointed to by 0(A6,A4.L). If an odd opcode is used (bit 0 is set), then the top six bytes of the maths stack are copied across to opcode-1(A6,A4.L) and A1 increased by 6, 'removing' the number from the maths stack (NOS becomes the new TOS).
If an even opcode is used (bit 0 is clear), then the six bytes stored at opcode(A6,A4.L) are copied across to the top of the maths stack (A1) is decreased by 6 creating a new TOS).

For **QA.OP** the operation code should be passed as a word in D0. For **QA.MOP** the operation codes are in a table of bytes pointed to by A3. The table is terminated by a zero byte.

Note: for the function EXP, D7 should be set to zero or an erroneous value will be returned.

The operation codes for the interpreter are as follows:

CODE		function	change to A1
$02	**qa.nint**	round fp to Nearest INTeger	+4
$04	**qa.int**	truncate fp to INTeger	+4
$06	**qa.nlint**	round fp to Nearest Long INTeger	+2
$08	**qa.float**	FLOAT integer	-4
$0A	**qa.add**	ADD (top of stack to next of stack)	+6
$0C	**qa.sub**	SUBtract (tos from nos)	+6
$0E	**qa.mul**	MULtiply (tos by nos)	+6
$10	**qa.div**	DIVide (tos into nos)	+6
$12	**qa.abs**	ABSolute value	0
$14	**qa.neg**	NEGate	0
$16	**qa.dup**	DUPlicate	-6
$18	**qa.cos**	COSine	0
$1A	**qa.sin**	SINe	0
$1C	**qa.tan**	TANgent	0
$1E	**qa.cot**	COTangent	0
$20	**qa.asin**	ArcSINe	0
$22	**qa.acos**	ArcCOSine	0
$24	**qa.atan**	ArcTANgent	0
$26	**qa.acot**	ArcCOTangent	0
$28	**qa.sqrt**	SQuare RooT	0
$2A	**qa.log**	Log (Natural)	0
$2C	**qa.l10**	Log base 10	0
$2E	**qa.exp**	Exponential	0
$30	**qa.pwrf**	raise to PoWeR (Floating point) (nos to power of tos)	+6

In addition, SMSQ and Minerva support the following function codes:

$01	**qa.one**	push constant one	-6
$03	**qa.zero**	push constant zero	-6
$05	**qa.n**	followed by a signed byte, to push FP -128 to 127	-6
$07	**qa.k**	plus a byte, nibbles select mantissa and adjust exponent Following byte values may be: **qa.pi180** $56 **qa.loge** $69 **qa.pi6** $79 **qa.ln2** $88-$100 **qa.sqrt3** $98-$100 **qa.pi** $A8-$100 **qa.pi2** $A7-$100	-6
$09	**qa.flong**	float a long integer	-2
$0D	**qa.halve**	TOS / 2	0
$0F	**qa.doubl**	TOS * 2	0
$11	**qa.recip**	1 / TOS	0
$13	**qa.roll**	(TOS)B, C, A => (TOS)A, B, C (roll third to top)	0
$15	**qa.over**	NOS	-6
$17	**qa.swap**	NOS <=> TOS	0
$25	**qa.arg**	arg(TOS,NOS)=a, solves TOS=k*cos(a) & NOS=k*sin(a)	+6
$27	**qa.mod**	sqrt(TOS^2+NOS^2)	+6
$29	**qa.squar**	TOS * TOS	0
$2F	**qa.power**	NOS ^ TOS, where TOS is a signed short integer	+2

Vector $11A

QA.RESRI

Reserve Room on Arithmetic Stack

Call parameters		Return parameters	
D1.L	nr. of bytes required	D1	???
D2		D2	???
D3		D3	???
A0		A0	preserved
A1	ptr to RI stack (rel. A6)	A1	???
A2		A2	preserved
A3		A3	preserved

Error returns:

IMEM out of memory [SMSQ]
none [QDOS]

All addresses passed to this routine must be relative to A6.

QA.RESRI is used to reserve space on the arithmetic stack (A6,A1).

Since not only the stack but the whole SuperBASIC area may move during the call, the arithmetic stack pointer should be saved in **BV_RIP(A6)**, whence it should be retrieved after the call has been completed.

Vector $112	SuperBASIC get Integer parameter(s)	SB.GTINT
$114	SuperBASIC get Floating point parameter(s)	**SB.GTFP**
$116	SuperBASIC get String parameter(s)	**SB.GTSTR**
$118	SuperBASIC get Long Integer parameter(s)	**SB.GTLIN**

Call parameters		Return parameters	
D1		D1	???
D2		D2	???
D3		D3.W	number of parameters fetched
D4		D4	???
D6		D6	???
A0		A0	???
A1		A1	ptr to RI stack (rel. A6)
A2		A2	???
A3	ptr to name table entry for 1st parameter (rel. A6)	A3	preserved
A4		A4	preserved
A5	ptr to name table entry for last parameter (rel. A6)	A5	preserved

Error returns:

standard, condition codes set

All addresses passed to these routines must be relative to A6.

These routines are used to get the values of actual parameters to SuperBASIC procedures or functions onto the arithmetic stack. Each routine assumes that all the parameters will be of the same type, as follows:

SB.GTINT	16-bit parameter
SB.GTFP	floating point
SB.GTSTR	string
SB.GTLIN	floating point: convert to 32-bit long integer

The values are returned in the order on the arithmetic stack (A6,A1) with the first parameter at the top (lowest address) of the stack.

The separator flags in the name table entries are smashed by this routine.

SB.INIPR

Vector $110

Initialise SuperBASIC procedures and functions

Call parameters		Return parameters	
D1		D1	preserved
D2		D2	???
		D3+	preserved
A0		A0	preserved
A1	pointer to proc/fn table	A1	???
		A2+	preserved

Error returns:

IMEM no room for table

SB.INIPR is used to link in a list of procedurs and functions to be added to the SuperBASIC name table. Once added, the functions can be called from SuperBASIC in the same way as the procedures and functions built into the ROM.

The structure of the proc/fn table is defined in the following form:

word	approximate number of procedures (see below)

for each procedure

word	pointer to routine - here
byte	length of name of procedure
characters	name of procedure
word	0
word	approximate number of functions (see below)

for each function

word	pointer to routine - here
byte	length of name of function
characters	name of function
word	0

The "approximate number" of procedures or functions is used to reserve internal table space. It can be calculated with the following formulae:

INT ((total number of characters used in procedures or functions + 6)/7)

The pointers to the routines are relative to the address of the program counter, e.g.

```
DC.W ENTRY-*
```

Vector $120 **SB.PUTP**

SuperBASIC put Parameter

Call parameters		Return parameters	
D1		D1	???
D2		D2	???
D3		D3	???
A0		A0	???
A1	ptr to value to be assigned (rel. A6)	A1	???
A2		A2	???
A3	ptr to name table entry (rel. A6)	A3	preserved

Error returns:

standard error code

All addresses passed to this routine must be relative to A6.

SB.PUTP assigns a value to be associated with an entry in the SuperBASIC name table. For details of the value to be assigned see section 9.5. A1 and A3 should be on word boundaries.

The type of the entity to be assigned (and hence its length) is determined by the type in the name table entry.

BV_RIP(A6) must point to the value to be returned (top of arithmetic stack). BV_RIP will be updated on return by SB.PUTP.

Vector $E6

UT.CSTR

Compare two strings

Call parameters		Return parameters	
D0.B	comparison type	D0.L	-1, 0 or +1
D1		D1	preserved
D2		D2	preserved
D3		D3	preserved
A0	base of string 0 (rel. A6)	A0	preserved
A1	base of string 1 (rel. A6)	A1	preserved
A2		A2	preserved
A3		A3	preserved
A6	base address register	A6	preserved

All addresses passed to this routine must be relative to A6.
D0 (and the status register) is set negative if the string at (A6,A0) is less than the string at (A6,A1) etc.

The string comparison routine used by the directory system, and the Basic interpreter, uses an extended interpretation of the value of a string and has four modes of operation.

Order of Strings
Since comparison may be used to sort strings into order as well as checking for equality or equivalence, the order must be well defined. A form of dictionary order is attempted - this will require to be modified for foreign character sets.

Space is the first character.
Punctuation is in ASCII order (except "." which is the last).
All punctuation is defined to be before all letters or digits (e.g. **A.** before **AA.**).
Optionally, embedded numbers may be taken in numerical order (e.g. **Case5A** before **Case10A**, and also **Case5.10** before **Case5.5**).
All digits or numbers are defined to be before all letters (e.g. **bat1** before **bath1**).
An upper case letter comes before the corresponding lower case letter but after the previous lower case letter (e.g. **Bath** is before **bath** but after **axe**).
Optionally, an upper case letter is treated as quivalent to a lower-case letter.

SPACE
!"#$%&'()*+,-/:;<=>?@[\]^_£{|}~ Copyright.
Digits or numbers
AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpQqRrSsTtUuVvWwXxYyZz
Foreign characters

Comparisons
The relationship of one string to another may be

equal all characters or numbers are the same or equivalent.
lesser the first part of the first string, which is different from the corresponding character in the second string, is before it in the defined order.
greater the first part of the first string, which is different from the corresponding character in the second string, is after it in the defined order.

Types of Comparison

Comparisons may be made directly on a character by character basis (type 0), or made ignoring the case of the letters (type 1), or made using the value of any embedded numbers (type 2), or both ignoring the case of letters and using the value of embedded numbers (type 3).

File and vaiable name comparisons use type 1.
Basic <, <=, =, >=, > and <> operators use type 2.
Basic == (equivalence) operator uses type 3.

UT.WERSY
UT.WERMS

Vector $CA write error message to channel 0
$CC write error message to given channel

Call parameters		Return parameters	
D0.L	error code	D0.L	preserved
D1		D1	preserved
D2		D2	preserved
D3		D3	preserved
A0	channel ID (UT.WERMS only)	A0	preserved
A1		A1	preserved
A2		A2	preserved
A3		A3	preserved

UT.WERMS should be called from user mode. If A0=0, it can be called in Supervisor mode.

These routines exist for writing simple messages to a channel. They are basic error message handlers which write a standard or device driver supplied error message to either the command channel 0, or else to a defined channel.

UT.WINT

Vector $CE

Write an integer to ASCII and sent it to the defined channel

Call parameters		Return parameters	
D1.W	integer parameter	D1	???
D2		D2	???
D3		D3	???
A0	channel ID	A0	preserved
A1		A1	???
A2		A2	preserved
A3		A3	preserved

Error returns:

All the usual IO

This routine ought usually to be called from user mode. It can be called in Supervisor mode if A0=0.

Vector $D0

UT.WTEXT

Send a message to a channel

Call parameters		Return parameters	
D1		D1	???
D2		D2	???
D3		D3	???
A0	channel ID	A0	preserved
A1	base of message	A1	???
A2		A2	preserved
A3		A3	preserved

Error returns:

All the usual IO

This routine ought usually to be called from user mode.

The message is in the form of a text string: number of characters (word) followed by the characters in ASCII. If a newline is required at the end of the message, this should be included in the message. If the channel is 0 then D3 will be returned 0, otherwise D3 will be returned to -1. In version V1.03 and earlier, D0 is set to the error return but is not tested so the condition codes will not be correct. As a special concession, interrupt servers and other supervisor mode routines can call these routines with A0=0. If the command channel is in use, they will attempt to use channel 1. This operation is not reommended, but it does seem to work!

Vectored Routines - numerical order with page reference

17.0 New Concepts - Things [EXT]

Things are general-purpose resources which may be used by any code in the system, either from device drivers or directly from programs. In principle a Thing may be shareable by a finite or "infinite" number of "users", or restricted to one user at a time. A run-time system will be infinitely shareable, a two-port serial chip may have two users, and so on. The operating system provides suitable facilities for adding, removing and using Things.

Things are kept in a linked list, each one being identified by a name which must be unique. A new thing is added by setting up a suitable linkage block and then calling the operating system routine to link it into the list: the new thing will be rejected if its name is not unique. The linkage block must be in the common heap so that it may be discarded correctly when the Thing is removed. Each Thing has a version ID which will be returned to any Job which uses the Thing: this may be the familiar ASCII number, e.g. "1.03", or a bit map of implemented facilities, e.g. %10000101.

A piece of code that wishes to use a Thing supplies the system routine with the name of the Thing, and any additional parameters the Thing itself may require: this is very similar to the IOSS open call, except that the result returned is an address, not an "ID". The meaning of this address depends on what the Thing is. If the call to use a Thing is successful, then a new entry is made in the Thing's "usage list", marking the Thing as used by the given Job.

A piece of code may "free" a given Thing either by an explicit call to do so, or, if it is a Job, by being removed. As the code may "own" more than one instance of a thing (e.g. two serial ports), parameters may be passed to the Thing's FREE code to signal which instance is to be discarded. If the owner is a Job which is being removed, a special "Forced FREE" routine is called. If a Thing is freed on behalf of another job, then that Job will be removed.

If a Thing is not in use it may be removed from the list by the system routine provided, and its linkage block discarded. An attempt to remove a Thing that is in use will cause an error, in which case its linkage block must not be discarded. A Thing may supply a "remove" routine to tidy itself up before removal - for instance, a parallel I/O port would be set to all inputs.

A routine is provided to "force remove" a Thing. If the Thing is in use, then all Jobs using it will also be removed (with the exception of the Job that is doing the forced remove, unless that Job is owned by a Job that is itself using the Thing). In this case the linkage block is automatically returned to the common heap.

Thing linkage format

Items from **TH_THING** onwards (inclusive) must be filled in by the initialsation code before a new thing is added with the **SMS.LTHG** routine.

TH_NXTTH	$00	long	points to NeXT THing linkage block
TH_USAGE	$04	long	USAGE list
TH_FRFRE	$08	long	code called when Force Remove FREes a thing
TH_FRZAP	$0c	long	code called when thing owner is removed *
TH_THING	$10	long	points to THING itself
TH_USE	$14	long	code to invoke to USE the thing, or 0
TH_FREE	$18	long	code to invoke to FREE the thing, or 0
TH_FFREE	$1c	long	code to Force FREE a thing, or 0
TH_REMOV	$20	long	code to tidy up before REMOVing a thing, or 0
TH_NSHAR	$24	byte	byte set if Thing Not SHAReable
TH_VERID	$26	long	version ID, e.g. "1.03" or %1011101
TH_NAME	$2a	string	NAME of thing

Thing header format

All offsets are relative to the address of the flag.

THH_FLAG	$00	4 bytes	flag signalling standard header: value "THG%"
THH_TYPE	$04	long	type of Thing: -1=the THING code itself 0=utility code (free format) 1=executable code 2=shared data (free format) 3=extension code (user mode) 4=extension code (supervisor mode) bit 24 is set if the set if the Thing has a list Things within it.

List of Things Header

THH_NEXT	$08	long	offset of next Thing in list (0 for last)
THH_EXID	$0c	long	extra ID

Executable Thing Header

THH_HDRS	$08	long	offset to start of header
THH_HDRL	$0c	long	size of header
THH_DATA	$10	long	dataspace
THH_STRT	$14	long	offset of start of code or 0 to start at (copy of) header

Extension Thing Header

THH_PDEF	$10	long	offset to parameter definitions (or 0)
THH_PDES	$14	long	offset to parameter descriptions
THH_CODE	$18		entry point for extension code - should exit with RTS

Different sorts of Thing

Things may take many forms, but it may be useful to mention a few "tricks" relating to specific ones here. In particular, the programmer who wishes to make use of Things must cater for the eventuality that his Thing will be removed, probably forcibly.

Things in ROM will often link themselves in at boot: it may be desirable to have a SuperBASIC procedure to re-link them if removed, but otherwise no special problems present themselves.

Thing loaded into the resident procedure area act in a very similar way to ROM Things, except that if removed there is wasted RAM where the Thing is loaded.

Things loaded into the Transient Program area as active or inactive Jobs can have the space used reclaimed when they are removed. There are two ways in which such a Thing can be removed, one is by a Thing call (RTHG or ZTHG) and the other is via a remove Job call (FRJB). The Thing remove code must ensure that if the Job is removed, the Thing goes away, and vice versa. This may be accomplished by ensuring that the Job owns the Thing linkage block, and that the Thing remove code (a) sets the Job's PC to some code which will cause it to remove itself, (b) sets the Job's priority to 127, and (c) releases it from any current suspension. Note that as the Thing remove code is called from supervisor mode, it must not itself remove the Job.

Things loaded into common heap are the easiest to deal with. The easiest case is where the Thing can be loaded into a suitably extended Thing linkage block, in which case no special code is required. If this is not possible, the Thing remove code must release the heap entry containing the Thing. While it is conceivable that the heap containing the Thing will be released by some outside agency without calling a Thing remove routine, any such action may be regarded as so incredibly hostile that no precautions need be taken against it. This contrasts with the "unexpected" removal of a Job, which may be regarded as a fairly normal occurrence.

Hardware Things will frequently have some code or workspace in one or other of the above areas of RAM. The same comments thus apply, with the extra requirement that the hardware be placed in a "safe" state when the Thing controlling it is removed. Ideally this safe state will be the same as that obtained by resetting the computer.

17.1 Extension Things

This chapter defines a standard mechanism for a procedure interface that can, in principle, provide extensions to any programming language.

The structure allows several related procedures to be stored in one Thing. This simplifies maintenance and reduces the system overheads.

Parameters are passed to the extension using conventions similar to the C programming language. The parameter list contains keys and values passed to the routine and pointers to more complex parameters. The parameter list itself should not be modified. Each extension can have its own definition of the parameter list: there is both a formal definition to provide automatic interfacing to high level languages, and an informal description to provide user help texts.

The interface provides for procedures only. If a procedure has one principal return parameter, this should be defined as the last parameter in the list. A high level language interface can then identify this easily if the extension procedure is called as a high level language function. Note that this is different from calling a high level language procedure as a function where the error return would be expected as the function value.

Extension procedures should not normally allocate memory for the return parameters, the call mechanism provides that the amount of memory available for a return parameter is either fixed by the parameter type or is specified for a particular call.

If a procedure requires to return a variable size parameter, with no limit on its size, and the space pre-allocated is not sufficient, then it should return the error ERR.BFFL and the parameter list must be defined in such a way that procedure may be re-called. In this case it is unlikely that an automatic interface from the high level language will be appropriate.

Th aim of this definition is not to provide a universal interface which will cover all eventualities, but to make the interface in the majority of cases automatic, while keeping the interface simple and efficient.

Extension Thing Header

All offsets are relative to the address of the flag.

thh_flag	$00	4 bytes	flag signalling standard header: value "THG%"
thh_type	$04	long	type of Thing: value $01000003
thh_next	$08	long	offset of next Thing in list (0 for last)
thh_exid	$0c	long	extension ID
thh_pdef	$10	long	offset to parameter definitions (or 0)
thh_desc	$14	long	offset to description
thh_code	$18		entry point for extension code - should exit with RTS

Level 1 Extension Parameter Definition

The parameters for a extension thing are defined as a table of words. Each word defines the type of parameter that is possible. The table is terminated by a zero word. In general, a single call value or key is denoted by a positive word, while a pointer to a parameter value is negative. The value -1 is used to delimit a group of repeated parameters. The value -character is used to start a "keyed" group of parameters. Because extra information on pointer parameters is passed to the extension procedure, these parameters can be allowed to be one of a list of possible types. Note that extension procedures with optional or repeated parameters may have ambiguous definitions. Ambiguous parameter definitions cannot be handled by general purpose interface code from a high level language, so that such routines will require individually coded interfaces.

The simplest parameters are call values or keys. The parameter definitions for these are all low value, positive words. The distinction between a key and a call value is that the former has a significance which is defined internal to the extension procedure, while that latter has a numerical value.

Call Values and Keys

thp.key	$0001	key
thp.char	$0004	character
thp.ubyt	$0008	unsigned byte
thp.sbyt	$000a	signed byte
thp.uwrd	$0010	unsigned word
thp.swrd	$0012	signed word
thp.ulng	$0020	unsigned long
thp.slng	$0022	signed long
thp..opt	12	bit set if parameter optional
thp..nnl	11	bit set if null parameter is negative (-1)

For parameters where the item in the parameter list is a pointer to a value, the situation is rather more complex. For each parameter, there may be a number of possibilities. The word in the list is formed by ORing all the possibilities together. There are bits that define that the parameter is a pointer and defines whether the parameter is call, return, updated or specified by the calling code.

Pointer Parameter Usage

thp..ptr	15	bit set for pointer parameter
thp..cal	14	bit set for call parameter
thp..ret	13	bit set for return parameter
thp.upd	$e000	updated parameter
thp.call	$c000	call parameter
thp.ret	$a000	return parameter
thp.ptr	$8000	call or return parameter (specified by calling code)

If the parameter is optional, then the optional bit should be set (or the word is ORed with the optional key value.

Optional Parameter

thp..opt	12	bit set if parameter is optional
thp.opt	$1000	optional

The parameter could be an array of given type with a standard header: note that the standard interface code will always allow a single value to be used in its place.

Array Parameter

thp..arr	11	bit set for array
thp.arr	$0800	array

To finish of the definition word, the values defining each of the possible types of parameter should be ORed with the word so far. Note that, provided there is at most one signed value possible, the values representing the parameter usage, option, array and types may be ADDed together rather than ORed. Note also that a you may not have both unsigned and signed values.

Parameter Types

thp..sgn	1	bit set if value is signed
thp..chr	2	bit set if character allowed
thp..byt	3	bit set if byte value allowed/required
thp..wrd	4	bit set if word value allowed/required
thp..lng	5	bit set if long value allowed/required
thp..str	8	standard string
thp..sst	9	sub-string
thp.char	$0004	character
thp.ubyt	$0008	unsigned byte
thp.sbyt	$000a	signed byte
thp.uwrd	$0010	unsigned word
thp.swrd	$0012	signed word
thp.ulng	$0020	unsigned long
thp.slng	$0022	signed long
thp.fp8	$0042	eight byte floating point
thp.str	$0100	string
thp.sstr	$0200	sub-string

Example Parameter Definitions

```
COPY
            dc.w      thp.call+thp.str        pointer to source file
            dc.w      thp.call+thp.str        pointer to destination file
            dc.w      0

SER_BUFF
            dc.w      thp.opt+thp.ulng        optional unsigned long
            dc.w      thp.opt+thp.ulng        optional unsigned long
            dc.w      0

PRT_USE$
            dc.w      thp.ret+thp.str         pointer to return string
            dc.w      0
```

Parameter List

For each parameter that is passed there is one or two long words in the parameter list. For a key it is just the key in a long word. The procedure itself will determine how much of the key is significant. For a call value, the value is in the least significant part of the long word, the rest of the long word is ignored. If a key or call parameter is marked as optional, then the interface code should provide a default value (normally zero or -1 depending on thp..nnl) if the parameter is missing

For a pointer there are eight bytes: two words followed by a long word. The first word specifies the usage of the parameter. If it was an optional parameter and it is missing, the value is 0. Otherwise thp..ptr and either or both thp..cal and thp..ret are set. The thp..arr bit will be set if the pointer is to an array. In addition, one of the lower bits must be set to define the type of parameter. The thp..sng and thp..key bits should be clear.

The next word is zero for most parameters, but for a return string it is the maximum space available, and for a call sub-string it is the length of the sub-string.

The next long word is the pointer to the parameter value (or array definition). If it is a missing optional parameter the value is ignored, but, for future compatibility, zero should be supplied.

A repeated group of parameters is prefaced by a long word with the number of repeats.

Defining Extension Things

Extension Things do not need to be written to strict rules. Since it can be assumed that the code calling the Extension Thing is fully aware of the requirements and behaviour of the Extension Thing, an Extension Thing can be any routine. It is, however, advantageous to be more strict than this. If the Extension Thing is defined with an unambiguous parameter definition, and it accepts a parameter list in the standard form described above, and it is clean to the extent of preserving all registers except d1 and a1 (meeting the SuperBASIC interpreter requirements for a6 and a7 as well), and it returns a standard error code (-ve) or escape code (+ve) or zero in d0, and it has at most one return parameter, then it will usually be possible to interface to the Extension Thing automatically.

The format of an Extension Thing does not allow more than a four character ID. This is to simplify access. It is up to the high level language itself to define a suitable name although the name in the informal description may be used.

One requirement of the definition of an Extension Thing is that it must be shareable.

Accessing Extension Things

Depending on the extent to which an Extension Thing is to be used, an application can either USE the Extension Thing during initialisation and save the address of the Extension Thing (and possibly the Thing linkage) or it can USE the Extension Thing as required and FREE it immediately afterwards. The latter is simpler, the former is more efficient for small, frequently used Extension Things.

When to Use Extension Things.

There are many ways of extending the SMS2 operating system. Using an Extension Thing is just one. There are two cases where it is appropriate to add an extension thing. The first is where the extension is provided to access some hardware dependent facility or other facility which is an optional extra. Provided that the Extension Thing has an unambiguous parameter definition and a clean interface, it should be possible to add such an extension to any high level language. The second case is where there is a facility which is likely to be required to be called from a number of languages and involves a considerable amount of code. In this case, it is not so important that the facility has either a unambiguous definition or a clean interface.

The SER_PAR_PRT extension things are good examples of the first. These are very simple extensions which are linked to the serial and parallel port drivers. The FILE_SELECT extension is a good example of the latter, this is a very complex, but useful procedure.

An Extension Thing may not be appropriate if the procedure is just a direct interface to a operating system facility (e.g. INK, PAPER, CLS etc.).

Thing Vectors

To enable the thing system to be used from user code under QDOS, which does not allow the TRAP #1 to be extended, versions 2.03 onwards of the HOTKEY System II add a strange Thing to the end of the Thing list. This Thing has the name THING and is not accessible using the Thing system and so may not be removed. The THING Thing is $18 bytes long.

THH_FLAG	$00	long	'THG%'
THH_TYPE	$04	long	-1
THH_ENTR	$08	long	absolute address of TH_ENTRY routine
THH_EXEC	$0C	long	absolute address of TH_EXEC routine

To find the THING Thing, pick up the pointer **SYS_LTHG** ($B8 on from the base of the system variables), and follow the linked list to the end. The last item in the list should be the THING Thing.

Hotkey Vectors

The Hotkey vectors are in the Hotkey Thing. These are available in all HOTKEY System II versions.

Thing Entry Points

TH_ENTRY
entry point is for calling from user mode: in SMS2 they are replaced by a TRAP #1, and the entry vectors added in with the rest. The parameters are exactly the same as for the SMS2 version, though. Under QDOS, all calls to SMS.ZTHG must be made in user mode, as must calls to FTHG on behalf of another Job.

TH_EXEC
This executes the code of an executable thing, setting the standard parameter string and opening a file for the job if required. It returns an error code in D0, and is called with D1 holding the owner ID, 0, or -1. The MSW of D2 should contain the priority of the job to be executed, and the LSW should contain the timeout. A0 must contain a pointer to the Thing name, A1 is a pointer to the parameter string.

Example of entries to the Thing Vector system:

```
; Jump to Thing Utility through HOTKEY System II
; Copyright 1989 Tony Tebby / Jochen Merz
; Note this only works if a HOTKEY System version 2.03 or later is present.
;
;               Entry                           Exit
;         d1    owner                           Job ID
;         d2    priority/timeout                preserved
;         a0    thing name                      preserved
;         a1    parameter string                preserved
;
;       Condition codes set
;
ut_thjmp
     move.l    a4,-(sp)
     move.l    d0,-(sp)
     moveq     #thh_entr,d0            ; thing vector required
     bsr.s     gu_thvec                ; get THING vector
     bne.s     gut_ex4                 ; there's nothing to jump to!
     move.l    (sp)+,d0
     jsr       (a4)                    ; do it
gut_exit
     move.l    (sp)+,a4
     tst.l     d0
     rts
gut_ex4
     addq.l    #4,sp                   ; skip operation
     bra.s     gut_exit
```

```
; Find Thing utilitiy vector of HOTKEY System II.
; Note this only works if a HOTKEY System version 2.03 or later is present.
;
;               Entry                   Exit
;         d0    vector required         error code
;         a4                            Thing Utility Vector
;
;       Error returns:  err.nimp                THING does not exist
;       Condition codes set
;
gu_thvec
     movem.l   d1-d3/d7/a0,-(sp)
     move.w    d0,d3
     moveq     #sms.info,d0             ; get system variables
     trap      #do.sms2
     move.w    sr,d7                    ; save current SR
     trap      #0                       ; into supervisor mode
     move.l    sys_thgl(a0),d1          ; this is the Thing list
     beq.s     thvec_nf                 ; empty list, very bad!
     move.l    d1,a0
thvec_lp
     move.l    (a0),d1                  ; get next list entry
     beq.s     th_found                 ; end of list? Should be THING!
     move.l    d1,a0                    ; next link
     bra       thvec_lp
thvec_nf
     moveq     #err.nimp,d0             ; THING does not exist
     bra.s     thvec_rt
th_found
     move.l    th_thing(a0),a0          ; get start of Thing
     cmp.l     #-1,thh_type(a0)         ; is it our special THING?
     bne.s     thvec_nf                 ; sorry, it isn't
     move.l    (a0,d3.w),a4             ; this is the vector we look for
thvec_rt
     move.w    d7,sr                    ; back into previous state
     movem.l   (sp)+,d1-d3/d7/a0
     tst.l     d0
     rts
```

The following example demonstrates how to create and link in a Thing. Two areas are allocated, one for the Thing contents, one for the Thing linkage. The contents may already be present in RAM or ROM/EPROM, but the linkage has to be in RAM. The demonstration Thing is a simple translation table.

```
     move.l    #8+264,d1                ; thh_flag+thh_type+tra_table
     bsr       demo_achp                ; allocate heap
     bne       demo_exit                ; failed!

     move.l    a0,-(sp)
     moveq     #$38,d1                  ; room for linkage
     bsr.s     demo_achp
     move.l    a0,a1                    ; the linkage
     move.l    (sp)+,a0                 ; that's the Thing address
     beq.s     demo_lact                ; linkage allocated
```

```
        move.l   d0,-(sp)                ; preserve error
        moveq    #sms.rchp,d0            ; second ACHP failed, return first
        trap     #do.sms2
        move.l   (sp)+,d0                ; return error to calling code
        bra.s    demo_exit

demo_lact
        lea      th_thing(a1),a2         ; fill in linkage
        move.l   a0,(a2)+                ; pointer to Thing
        clr.l    (a2)+                   ; no special use
        clr.l    (a2)+                   ; and no special free
        clr.l    (a2)+                   ; and no special force free
        clr.l    (a2)+                   ; also no special remove code
        clr.w    (a2)+                   ; it's shareable
        move.l   #'1.00',(a2)+           ; version
        move.w   #$09,(a2)+              ; length of name
        move.l   #'Tran',(a2)+           ; name
        move.l   #'slat',(a2)+           ; name
        move.b   #'e',(a2)

        move.l   #'THG%',(a0)+           ; standard Thing flag
        move.l   #2,(a0)+                ; Type data
        move.w   #$4afb,(a0)+            ; now fill in TRA table
        move.w   #6,(a0)+                ; first offset
        move.w   #262,(a0)+              ; second offset
        moveq    #0,d0
demo_loop
        move.b   d0,(a0)+                ; fill in 1 to 1 translation
        addq.b   #1,d0                   ; for all 256 characters
        bne.s    demo_loop
        clr.w    (a0)                    ; end word

        moveq    #thh_entr,d0            ; thing vector required
        bsr.s    gu_thvec                ; get THING vector
        bne.s    demo_exit               ; there's nothing to jump to!
        lea      th_name(a1),a0          ; name
        moveq    #sms.zthg,d0
        jsr      (a4)                    ; zap it (in case, it exists)
        moveq    #sms.lthg,d0            ; link it
        jsr      (a4)
demo_exit
        rts

demo_achp
        moveq    #sms.achp,d0            ; allocate heap
        moveq    #0,d2                   ; for system
        trap     #do.sms2
        tst.l    d0                      ; failed?
        rts
```

Thing-supplied code

More complex Things may need to provide code to be invoked when the Thing is used, freed and removed. The addresses of any such routines must be filled in in the Thing linkage block before the **SMS.LTHG** routine is called to add the Thing into the list. If a routine address is zero then the internal routines will be used - these cater for the most frequent case of an infinitely-shareable thing. All the following routines will be called in Supervisor mode, and should end with an **RTS** instruction. Note that as a result of this, they must not call any of the non-atomic TRAPs.

Thing use routine **TH_USE**

Call parameters		Return parameters	
D1	Job ID	D1	???
D2	additional parameter	D2	additional result
D3	additional parameter	D3	???
		D4+	???
A0		A0	usage block
A1	Thing linkage block	A1	???
A2	additional parameter	A2	additional result
		A3-A5	???
A6	system variables	A6	???

Error returns:

D0 and the status register must be set

This routine will be called from within the **SMS.UTHG** routine to generate a non-standard usage block. If the Thing cannot be used, or the parameters supplied are incorrect, then an error may be returned instead. The usage block pointed to by A0 should be a standard heap entry as allocated by the **MEM.ACHP** vector (A0 points to the header, not the "usable memory), of which the first $18 bytes (heap header + 8) are reserved for the use of the operating system. Additional parameters passed by the calling code in D2/D3/A2 are unchanged, and results may be returned to the calling code in D2 and A2.

Thing free routine		TH_FREE
Call parameters		Return parameters
D1 Job ID		D1 ???
D2 additional parameter		D2 additional result
D3 additional parameter		D3 ???
		D4+ ???
A0 usage block		A0 usage block to unlink
A1 Thing linkage block		A1 ???
A2 additional parameter		A2 additional result
		A3-A5 ???
A6 system variables		A6 ???

Error returns:

It is assumed that this routine always succeeds

This routine will be called from within the **SMS.FTHG** routine to remove a non-standard usage block. A0 points to the first usage block in the Thing's usage list that is owned by the Job specified - depending on the passed parameters this may or may not be the usage block to be removed. When the correct usage block has been found, any internal tidying up should be performed, and the block should be returned to the heap.
Its address should then be returned so that it may be unlinked from the usage list.

Thing forced free routine		TH_FFREE
Call parameters		Return parameters
		D1+ ???
A0 usage block		A0 preserved
A1 Thing linkage block		A1 ???
		A2-A5 ???
A6 system variables		A6 ???

Error returns:

It is assumed that this routine always succeeds

This routine will be called from within the operating system when the Job that owns the usage block pointed to is force removed. One call will be made for each usage block in the Thing's usage list. As with the standard free routine, the usage block should be returned to the heap by this routine.

Thing remove routine — TH_REMOV

Call parameters		Return parameters	
		D1+	???
A0		A0	???
A1	Thing linkage block	A1	???
		A2-A5	???
A6	system variables	A6	???

Error returns:

It is assumed that this routine always succeeds

This routine is called from the **SMS.RTHG** and **SMS.ZTHG** routines when a Thing is to be removed entirely. It should ensure that everything associated with the Thing is in a "safe" state: this would include setting hardware to a suitable state, freeing any extra heap entries and soon. It must also return the Thing linkage block to the heap.

17.2 The HOTKEY System II [EXT]

The concept and function of HOTKEY System II is not described here, there are many manuals available how to use it (from the end-user's point of view). This section explains how to use the HOTKEY System II from machine code.

The HOTKEY System II is an exclusive Thing, so the code which uses the Thing should free it preferably very soon. There should be a timeout of about 2 seconds, otherwise the use-routine should give up. A sample how to get the HOTKEY linkage block (which is necessary for all routines using the HOTKEY System II) is

```
     moveq    #sms.uthg,d0   ; we want to use
     moveq    #sms.myjb,d1   ; for me
     moveq    #127,d3        ; wait for use
     lea      hk_thing,a0    ; name of thing
     trap     #do.sms2       ; do it
     move.l   a1,a3          ; the HOTKEY linkage must be in A3
     tst.l    d0
     rts

hk_thing  dc.w 6,'Hotkey'
```

The HOTKEY linkage contains vectors for the various facilities of the HOTKEY System II:

hk.fitem	$0014	find item
hk.crjob	$0018	hotkey create job
hk.kjob	$001c	hotkey kill job
hk.set	$0020	hotkey set
hks.off	-1	turn off
hks.on	0	turn on
hks.rset	1	reset
hks.set	2	set
hk.remov	$0024	hotkey remove
hk.do	$0028	hotkey do
hk.stbuf	$002c	hotkey stuff buffer
hk.gtbuf	$0030	hotkey get buffer (d0=0 current -1 prev)
hk.guard	$0034	hotkey guardian / grabber (2.04 onwards)

To call a routine, get the vector and JSR it. To stuff a string into the Stuffer Buffer, get the HOTKEY linkage, load the registers, then call the routine:

```
     move.l   hk.stbuf(a3),a2    ; get vector
     jsr      (a2)               ; call it
```

Finally, free the HOTKEY system as soon as possible!

HK.FITEM

Find a HOTKEY item

Call parameters		Return parameters	
D1		D1.w	HOTKEY
D2		D2.w	HOTKEY number (-ve if off)
		D3+	preserved
A1	HOTKEY item name	A1	ptr to HOTKEY item
A2		A2	preserved
A3	linkage block	A3	preserved

Error returns:

ITNF item not found

This routine finds a hotkey item given a pointer to a name or key string and removes references from the hotkey table and pointer list.

HK.CRJOB

Create the HOTKEY job

Call parameters		Return parameters	
		D1+	preserved
A3	linkage block	A3	preserved

Error returns:

all system errors related to jobs

HK.KJOB

Kill the HOTKEY job

Call parameters		Return parameters	
		D1+	preserved
A3	linkage block	A3	preserved

Error returns:

always succeeds

HK.SET

Set or reset a HOTKEY

Call parameters		Return parameters	
D0.b	op: -1=off, 0=on, +1=reset, +2=set		
D1.w	new key (reset, set; d0=+1 or +2)	D1.w	HOTKEY
		D2+	preserved
A1	ptr to item (set), ptr to key or name (off, on, reset)		
A2		A2	preserved
A3	linkage block	A3	preserved

Error returns:

FDNF hotkey not found (off, on, reset)
FDIU hotkey in use (reset, set)

This routine can reset the state of a Hotkey to on or off. It can reset the Hotkey character for a current hotkey. It can set a new Hotkey item.

HK.REMOV

Remove HOTKEY item

Call parameters		Return parameters	
		D1+	preserved
A1	pointer to item name	A1	???
A2		A2	preserved
A3	linkage block	A3	preserved

Error returns:

ITNF item not found

Remove hotkey ITEM, this always removes the key and pointer. For defined stuffer items, it also returns the ITEM to the common heap. For nop, execute file or pick, it also returns the ITEM to the common heap. For executable Thing items, it also returns the ITEM and the THING.

HK.DO

"DO" a HOTKEY item

Call parameters		Return parameters	
		D1+	preserved
A1	pointer to HOTKEY item	A1	preserved
A2		A2	preserved
A3	linkage block	A3	preserved
A6	bottom limit of stack (for pick/wake job)	A6	preserved

Error returns:

ITNF item not found

HK.STBUF

Set a string in the stuffer buffer

Call parameters		Return parameters	
D2.w	number of characters to stuff	D2.w	preserved
		D3+	preserved
A1	pointer to characters	A1	preserved
A2		A2	preserved
A3	linkage block	A3	preserved

Error returns:

always succeeds

Set a new string in the stuffer buffer. It does not stuff a new string if this is the same as the previous string.

HK.GTBUF

Get stuffer buffer contents

Call parameters		Return parameters	
D0.b	key: 0=current, -1=previous string		
D2.w		D2.w	length of string
		D3+	preserved
A1		A1	pointer to characters
A2		A2	preserved
A3	linkage block	A3	preserved

Error returns:

always succeeds

HK.GUARD

Open and clear guardian window

All registers preserved.

Opens and clears guardian window. The definition must immediately follow the call. Then, if next word is non-zero, grab all but memory specified.

The HOTKEY Item

The HOTKEY Item has two words identifying the HOTKEY, followed by a pointer and then the name. The name is a composite which can include a considerable variety of information about the HOTKEY.

hki_id	$0000	word	hotkey id
hki.id	'hi'		
hki_type	$0002	word	hotkey item type
hki..trn	0		bit set if item is transient thing
hki.llrc	-8		last line recall
hki.stpr	-6		stuff kbd with prevous string from buffer
hki.stbf	-4		stuff keyboard queue from buffer
hki.stuf	-2		stuff keyboard queue with string
hki.cmd	0		pick SuperBASIC and stuff command
hki.nop	2		just do code
hki.xthg	4		execute thing
hki.xttr	5		as hki.xthg but thing is transient
hki.xfil	6		execute file
hki.pick	8		pick job
hki.wake	10		pick and wake job (execute thing if fails)
hki.wktr	11		as hki.wake but thing is transient
hki.wkxf	12		pick and wake job (execute file if fails)
hki_ptr	$0004	long	pointer to (preprocessing) code, stuff buffer
hki_name	$0008	string	item name

For last line recall and stuffing the keyboard queue from the buffer, the name is absent or irrelevant. For stuffing a string or command, the name is the string or command.

If the Hotkey can execute a Thing or file, the item name contains the Thing name or filename. The Thing name or filename may be followed by a semicolon then the parameter string enclosed by braces.

If there is a Wake or Job name which is different from the filename, this will be at the end of the item name, separated by an exclamation mark (Wake name) or comma (Job name).

17.3 The Button Frame [EXT]

The concept of the Button Frame (built into QPAC II) described here. Owners of the Button Frame software have to be owners of QPAC II, resulting in being owner of the manual. This section explains how to use the Button Frame from machine code.

The Button Frame is a shareable Thing. Every Job trying to place a Button in the Button Frame requests a position by trying to use the Button Frame. When the job is removed, the position in the Button Frame is automatically freed by the Thing system. If the Job does not already have an allocation in the frame, or a new allocation is required, the use routine looks for a hole in the Button Frame and, if successful, allocates a usage block with the Size and Position of the button. If the Job does have an allocation, and it is big enough, then the allocation is unaltered. If it is not big enough, then the button is re-allocated.

The name of the Thing is

```
dc.w 12,'Button Frame'
```

BT_USE

Use the Button Frame

Call parameters		Return parameters	
D1	user Job ID	D1	Job ID
D2.l	Button Size	D2.l	Button Origin
D3.l	0 new alloc, -ve for re-allocate	D3	version
A0	ptr to Thing Name	A0	preserved
A1		A1	pointer to Thing
A2		A2	pointer to Thing linkage
A3		A3	???

Error returns:

ORNG no room in button frame
FEX re-allocated

After a Button has been woken, the Button Frame should be freed unless the position of the Button should be kept for the next sleep. The Free routine finds the appropriate usage block, then frees the item in the button frame and throws the usage block away. If it cannot find the right usage block, it throws the first one away.

BT_FREE

Free the Button Frame

Call parameters		Return parameters	
D1	user Job ID	D1+	preserved
A0	ptr to name of Thing	A0	preserved
A1		A1	preserved
A2	base of usage block or 0 for 1st one	A2	???
		A3+	all preserved

Error returns:

always successful

18.0 Keys

The following section contain keys for various features of Qdos. These keys provide a definition for several of the data structures within Qdos.

18.1 Error keys

The following keys indicate error messages already defined in the system. A large positive error code is taken as the address of a user-supplied error message with bit 31 set. See the Concepts manual for a fuller description of the way in which these are used by the procedures built into SuperBASIC.

err.nc	-1	operation Not Complete
err.ijob	-2	Invalid JOB id
err.imem	-3	Insufficient MEMory
err.orng	-4	parameter Outside permitted RaNGe (c.f. err.ipar)
err.bffl	-5	BuFfer FuLl
err.ichn	-6	Invalid CHaNnel id
err.fdnf	-7	File or Device Not Found
err.itnf	-7	ITem Not Found
err.fex	-8	File already EXists
err.fdiu	-9	File or Device or In Use
err.eof	-10	End Of File
err.drfl	-11	DRive FuLl
err.inam	-12	Invalid file, device or thing name
err.trns	-13	TRaNSmission error
err.prty	-13	PaRiTY error
err.fmtf	-14	ForMaT drive Failed
err.ipar	-15	Invalid PARameter (c.f. err.orng)
err.mchk	-16	file system Medium CHecK failed
err.iexp	-17	Invalid EXPression
err.ovfl	-18	arithmetic OVerFLow
err.nimp	-19	operation Not IMPlemented
err.rdo	-20	ReaD Only permitted
err.isyn	-21	Invalid SYNtax
err.rwf	-22	Read or Write Failed [SMS2]
err.noms	-22	No error message [SMSQ]
err.accd	-23	Access denied [SMSQ]

18.2 System variables

The following list gives the offset of each system variable from the base of the system variables (whose position can be found using the **SMS.INFO** trap), together with the length of the variable.

sys_idnt	$0000	long	system variables identifier
sysid.ql	$d2540000		QL (QDOS) system variable identifier
sysid.at	'S2AT'		SMS Atari system variable identifier
sysid.sq	'SMSQ'		SMSQ identifier
sysid.th	$dc010000		Thor (ARGOS) system variable identifier

The following variables are the pointers which define the current state of the Qdos memory map.

sys_chpb	$0004	long	Common HeaP Base
sys_chpf	$0008	long	Common HeaP Free space pointer
sys_fsbb	$000c	long	Filing system Slave Block area Base
sys_sbab	$0010	long	'QL SuperBASIC' Area Base
sys_tpab	$0014	long	Transient Program Area Base
sys_tpaf	$0018	long	Transient Program Area Free space pointer
sys_rpab	$001c	long	Resident Procedure Area Base
sys_ramt	$0020	long	user RAM Top (+1)
sys_mxfr	$0024	long	Maximum return from free memory call [SMS]
sys_rtc	$0028	long	real time (seconds) [SMS]
sys_rtcf	$002c	word	real time fractional, count down [SMS]
sys_rand	$002e	word	RANDom number
sys_pict	$0030	word	Polling Interupt CounT
sys_dtyp	$0032	byte	Display TYPe (0=normal, 1=TV 625, 2=TV 525)
sys_dfrz	$0033	byte	Display FRoZen (T or F)
sys_qlmr	$0034	byte	QL Master chip Register value (Copy of MC_STAT)
sys_qlir	$0035	byte	QL Interrupt Register value (Copy of PC_INTR)
sys_rshd	$0036	byte	true to reschedule [SMS]
sys_nnnr	$0037	byte	Network Node NumbeR

The following system variables are pointers to the list of tasks and drivers.

sys_exil	$0038	long	EXternal Interrupt action List
sys_poll	$003c	long	POLled action List
sys_shdl	$0040	long	ScHeDuler loop action List
sys_iodl	$0044	long	IO Driver List
sys_fsdl	$0048	long	Filing System Driver List
sys_ckyq	$004c	long	Current Keyboard Queue
sys_ertb	$0050	long	Exception Redirection Table Base

The following system variables are pointers to the resource management tables. The slave block tables have 8 byte entries, whilst the others have 4 byte entries.

sys_sbrp	$0054	long	Slave Block Running Pointer
sys_sbtb	$0058	long	Slave Block Table Base
sys_sbtt	$005c	long	Slave Block Table Top

sys_jbtg	$0060	word	next JoB TaG
sys_jbtp	$0062	word	highest JoB in table (ToP one)
sys_jbpt	$0064	long	current JoB PoinTer
sys_jbtb	$0068	long	JoB Table Base
sys_jbtt	$006c	long	JoB Table Top
sys_chtg	$0070	word	next CHannel TaG
sys_chtp	$0072	word	highest CHannel in table (ToP one)
sys_chpt	$0074	long	last checked CHannel PoinTer
sys_chtb	$0078	long	CHannel Table Base
sys_chtt	$007c	long	CHannel Table Top
sys_frbl	$0080	long	FRee Block List (to be returned to common heap) [SMS]
sys_tsdd	$0084	byte	Thor flag [THOR only]

The following variables contain information about how to treat the keyboard, and about other aspects of the IPC and serial port communications.

sys_caps	$0088	word	CAPS lock (0 if off, msbyte set if on)
sys_lchr	$008a	word	Last CHaRacter (for auto-repeat)
sys_rdel	$008c	word	Repeat DELay (20ms units)
sys.rdel	25		
sys_rtim	$008e	word	Repeat TIMe (20ms units)
sys.rtim	2		

sys_rcnt	$0090	word	Repeat CouNTer (decremented every 20ms)
sys_swtc	$0092	word	SWiTch queues Character
sys_qlbp	$0096	byte	QL BeePing
sys_brk	$0097	byte	set by keyboard break [SMSQ]
sys_ser1	$0098	long	receive channel 1 queue address [QL]
sys_ser2	$009c	long	receive channel 2 queue address [QL]
sys_tmod	$00a0	byte	ZX8302 transmit mode (includes baudrate) (copy of PC_TCTRL) [QL]
sys_ptyp	$00a1	byte	PRoCeSsor type $00=68000/8, $30=68030 etc. [SMSQ]
sys.mtyp	$1e		machine ID bits
sys.immu	$01		internal MMU
sys.851m	$02		68851 MMU
sys.ifpu	$04		internal FPU
sys.88xf	$08		68881 68882 FPU
sys_csub	$00a2	long	subroutine to jump to on capslock
sys_stmo	$00a6	word	serial xmit timeout [QL]
sys_dmiu	$00a6	byte	DMA in use [SMS2, ST, SMSQ]
sys_mtyp	$00a7	byte	Machine TYPe / emulator type [SMS,ST]
sys.mblt	+1		Blitter fitted [SMSQ, ST]
sys.herm	+1		Hermes fitted [SMSQ, QL]
sys.mst	$00		ordinary ST
sys.mstr	$02		Mega ST or ST with RTC
sys.msta	$04		Stacy
sys.mste	$06		ordinary STE
sys.mmste	$08		Mega STE
sys.mgold	$0a		Gold card
sys.msgld	$0c		SuperGold card
sys.mfal	$10		Falcon
sys.mtt	$18		TT
sys.mqxl	$1c		QXL
sys.mdsp	%11100000	display type mask	
sys.mfut	%00000000	Futura emulator or none	
sys.mmon	%00100000	Monochrome monitor	
sys.mext	%01000000	Extended 4 Emulator	
sys.mvme	%10000000	QVME emulator	
sys.mvga	%11000000	VGA	
sys_stmv	$00a8	word	value of serial timeout (1200/baud+1, i.e. 11=75 bps, 5=300 bps, 3=600 bps, 2=1200 bps, 1=2400 bps+) [QL]
sys_polf	$00a8	word	polling frequency [SMSQ]
sys.polf	50		... assumed polling frequency
sys_cfst	$00aa	word	flashing cursor status
sys_prgd	$00ac	long	pointer to PRoGram Default [EXT]
sys_datd	$00b0	long	pointer to DATa Default [EXT]
sys_dstd	$00b4	long	pointer to DeSTination Default [EXT]
sys_thgl	$00b8	long	pointer to THinG List [EXT]
sys_psf	$00bc	long	Primary stack frame pointer [SMSQ]
sys_200i	$00c0	byte	200 Hz in service [SMSQ]
sys_50i	$00c1	byte	50 Hz in service [SMSQ]
sys_10i	$00c2	byte	10 Hz in service [SMSQ]
sys_plrq	$00c3	byte	poll requested (-ve for request) [SMSQ]
sys_clnk	$00c4	long	pointer to console linkage [SMSQ]
sys_castat	$00c8	byte	-1 cache on, +1 instruction cache temp off [SMSQ]
sys_casup	$00c9	byte	cache suppressd timer [SMSQ]
sys.casup	$2	byte	1 full tick
sys_iopr	$00ca	word	IO priority [SMSQ]
sys_cbas	$00cc	long	current basic (copy of sys_jbpt) [SMSQ]
sys_fpu	$00d0	16 bytes [SMSQ]	

sys_prtc	$00e0	byte	set if real time clock protected [SMSQ]
sys_pmem	$00e1	byte	memory protection level [SMSQ, ST]
sys_slug	$00e2	word	slug level [SMSQ]
sys_mdrn	$00ee	byte	which mdv drive is running? [QL]
sys_mdct	$00ef	byte	mdv run-up run-down counter [QL]
sys_mdid	$00f0	8*byte	drive ID*4 of each microdrive [QL]
sys_mdst	$00f8	8*byte	status: 0=no pending ops [QL]
sys_fsdd	$0100	16*long	pointers to Filing System Drive Definitions
sys_fsdt	$0140		Filing System drive Definition table Top
sys.nfsd	$10		max Number of Filing System Drive definitions
sys_fsch	$0140	long	linked list of Filing System CHannel blocks
sys_xact	$0144	byte	set if XLATE active [QDOS V1.10+, not SMS2]
sys_xtab	$0146	long	pointer to XLATE table [QDOS V1.10+, not SMS2]
sys_erms	$014a	long	pointer to (QDOS) error message table [QDOS V1.10+, not SMS2]
sys_mstab	$014e	long	pointer to (SMSQ) message table [SMSQ]
sys_taskm	$0154	4 long	used by Taskmaster - conflicts with
sys_turbo	$0160	long	used by Turbo
sys_qsound	$0164	long	used by QSound
sys_ldmlst	$0168	long	language dependent module list [SMSQ]
sys_lang	$016c	word	current language [SMSQ]
sys_vers	$0170	long	operating system version [SMSQ]
sys_top	$0180		TOP of system vars - bottom of Supervisor Stack

The following area, between $180 and $480 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.

18.3 SuperBASIC Variables

bv_start	$00		start of pointers
bv_bfbas	$00	long	buffer base
bv_bfp	$04	long	buffer running pointer
bv_tkbas	$08	long	token list
bv_tkp	$0c	long	
bv_pfbas	$10	long	program file
bv_pfp	$14	long	
bv_ntbas	$18	long	name table
bv_ntp	$1c	long	
bv_nlbas	$20	long	name list
bv_nlp	$24	long	
bv_vvbas	$28	long	variable values
bv_vvp	$2c	long	
bv_chbas	$30	long	channel table
bv_chp	$34	long	
bv_rtbas	$38	long	return table
bv_rtp	$3c	long	
bv_lnbas	$40	long	line number table
bv_lnp	$44	long	
bv_chang	$48		change of direction marker
bv_btp	$48	long	backtrack stack during parsing
bv_btbas	$4c	long	
bv_tgp	$50	long	temporary graph stack during parsing
bv_tgbas	$54	long	
bv_rip	$58	long	arithmetic stack
bv_ribas	$5c	long	
bv_ssp	$60	long	system stack (real one!)
bv_ssbas	$64	long	
bv_endpt	$64		end of pointers
bv_linum	$68	word	current line number
bv_lengt	$6a	word	current length
bv_stmnt	$6c	byte	current statement on line
bv_cont	$6d	byte	continue ($80) or stop (0) processing
bv_inlin	$6e	byte	processing in-line clause or not loop (1), other ($ff) or off (0)
bv_sing	$6f	byte	single line execution on ($ff) or off (0)
bv_index	$70	word	name table row of last in-line loop index read
bv_vvfre	$72	long	first free space in variable value table
bv_sssav	$76	long	save sp for out/mem to go back to
bv_rand	$80	long	random number
bv_comch	$84	long	command channel

bv_nxlin	$88	word	which line number to start after
bv_nxstm	$8a	byte	which statement to start after
bv_comln	$8b	byte	command line save ($ff) or not (0)
bv_stopn	$8c	word	which stop number set
bv_edit	$8e	byte	program has been edited ($ff) or not (0)
bv_brk	$8f	byte	there has been a break (0) or not ($80)
bv_unrvl	$90	byte	need to unravel ($ff) or not (0)
bv_cnstm	$91	byte	statement to CONTINUE from
bv_cnlno	$92	word	line to CONTINUE from
bv_dalno	$94	word	current DATA line number
bv_dastm	$96	byte	current DATA statement number
bv_daitm	$97	byte	next DATA item to read
bv_cnind	$98	word	in-line loop index to CONTINUE with
bv_cninl	$9a	byte	in-line loop flag for CONTINUE
bv_lsany	$9b	byte	whether checking list ($ff) or not (0)
bv_lsbef	$9c	word	invisible top line
bv_lsbas	$9e	word	bottom line in window
bv_lsaft	$a0	word	invisible bottom line
bv_lenln	$a2	word	length of window line
bv_maxln	$a4	word	max nr of window lines The 2 words immediately following this will be overwritten on changing lenln and maxln
bv_totln	$a6	word	nr of window lines so far
bv_auto	$aa	byte	whether AUTO/EDIT on ($ff) or off (0)
bv_print	$ab	byte	print fromprtok ($ff) or leave in buffer (0)
bv_edlin	$ac	word	line number to edit next
bv_edinc	$ae	word	increment on edit range
bv_tkpos	$b0	long	pos of A4 in tklist on entry to PROC
bv_ptemp	$b4	long	temp pointer for GO_PROC
bv_undo	$b8	byte	undo rt stack IMMEDIATELY then redo procedure
bv_arrow	$b9	byte	down ($ff) or up ($01) or no (0) arrow
bv_lsfil	$ba	word	fill window when relisting at least to here
bv_wrlno	$bc	word	when error line number [QDOS V1.10+]
bv_wrstm	$be	byte	when error statement [QDOS V1.10+]
bv_wrinl	$bf	byte	when error in-line ($ff) or not (0) [QDOS V1.10+]
bv_wherr	$c0	byte	processing when error ($80) or not (0) [QDOS V1.10+]
bv_error	$c2	long	last error code [QDOS V1.10+]
bv_erlin	$c6	word	line number of last error [QDOS V1.10+]
bv_wvnum	$c8	word	number of watched (WHEN) variables [QDOS V1.10+]
bv_wvbas	$ca	long	base of WHEN variable table wrt VVBAS [QDOS V1.10+]
bv_end	$100		

18.4.1 Offsets on BASIC Channel Definitions

The following section gives the format of an entry in the SuperBASIC channel table. These entries can be monitored or modified by user-defined SuperBASIC procedures which need to have a channel attached using a '#n' construct.

ch.id	$00	long	channel ID
ch.ccpy	$04	float	current cursor position, y
ch.ccpy	$0a	float	current cursor position, x
ch.angle	$10	float	turtle angle
ch.pen	$16	byte	pen status (0 is up, 1 is down)
ch.chpos	$20	word	character position on line
ch.width	$22	word	width of line in characters
ch.spare	$24		.. spare ..
ch.lench	$28		length of channel definition block

18.4.2 BASIC Token Values

The following section defines the token values used for the internal storage of a SuperBASIC program.

tkb.space	$80	spaces in the listing - two bytes: token, count
tkw.keyw	$81	all sorts of keywords:
tkw.end	$8101	END
tkw.for	$8102	FOR
tkw.if	$8103	IF
tkw.rep	$8104	REPeat
tkw.sel	$8105	SELect
tkw.when	$8106	WHEN
tkw.def	$8107	DEFine
tkw.proc	$8108	PROCedure
tkw.fn	$8109	FuNction
tkw.go	$810a	GO
tkw.to	$810b	TO
tkw.sub	$810c	SUB
tkw.err	$810e	ERRor
tkw.rest	$8111	RESTORE
tkw.next	$8112	NEXT
tkw.exit	$8113	EXIT
tkw.else	$8114	ELSE
tkw.on	$8115	ON
tkw.ret	$8116	RETurn
tkw.rmdr	$8117	REMAINDER
tkw.data	$8118	DATA
tkw.dim	$8119	DIM
tkw.loc	$811a	LOCal
tkw.let	$811b	LET
tkw.then	$811c	THEN
tkw.step	$811d	STEP
tkw.rem	$811e	REMark
tkw.mist	$811f	MISTake

tkb.odds	$84	all sorts of separators:
tkw.lequ	$8401	(LET) =
tkw.coln	$8402	:
tkw.hash	$8403	#
tkw.comma	$8404	,
tkw.lpar	$8405	(
tkw.rpar	$8406	)
tkw.lbrc	$8407	{
tkw.rbrc	$8408	}
tkw.space	$8409	space (significant)
tkw.eol	$840a	end of line
tkb.oper	$85	all sorts of operators:
tkw.plus	$8501	+
tkw.minus	$8502	-
tkw.mulf	$8503	*
tkw.divf	$8504	/
tkw.ge	$8505	>=
tkw.gt	$8506	>
tkw.apeq	$8507	==
tkw.eq	$8508	=
tkw.ne	$8509	<>
tkw.le	$850a	<=
tkw.lt	$850b	<
tkw.bor	$850c	||
tkw.band	$850d	&&
tkw.bxor	$850e	^^
tkw.power	$850f	^
tkw.cnct	$8510	&
tkw.or	$8511	OR
tkw.and	$8512	AND
tkw.xor	$8513	XOR
tkw.mod	$8514	MOD
tkw.div	$8515	DIV
tkw.instr	$8516	INSTR
tkw.neg	$8601	negate
tkw.pos	$8602	positive!!
tkw.bnot	$8603	~~
tkw.not	$8604	~
tkb.name	$8800	name
		The name token is followed by a word index to the name table
tkw.quote	$8b22	string delimited by "quotes"
tkw.apost	$8b27	string delimited by 'apostrophes'
tkw.text	$8c00	text (after REMark)
		The string and text tokens are followed by a word (nr. of chars) and the characters (with a pad byte if odd)
tkb.lno	$8d00	line number (word)
tkb.seps	$8e	all sorts of formatting separators:
tkw.scoma	$8e01	separator comma
tkw.scoln	$8e02	semicolon
tkw.bslsh	$8e03	backslash
tkw.bar	$8e04	bar
tkw.sto	$8e05	separator TO

A number constant is represented by $Feeemmm with $0eeemmmm (eee: exponent, mmmm: mantissa) being the actual floating point number.

18.5 Job Header and Save Area Definitions

The location of the job table can be found by looking at the system variables **SYS_JBTB** and **SYS_JBTT**. Each entry in the table is a longword pointing to a block of $68 bytes in the format given here.

jcb_len *	$0000	long	LENgth of job in tpa
jcb_strt	$0004	long	STaRT address
jcb_ownr	$0008	long	OWNeR of this job
jcb_rflg	$000c	long	pointer to job Released FLaG (cleared on release)
jcb_tag *	$0010	word	job TAG (set by MT.CJOB)
jcb_pacc	$0012	word	Priority ACCumulator set to zero when the job is executing, incremented on each scheduler call if the job is active but not executing
jb_pinc	$0013	byte	priority increment [QL] the actual priority of the job, set to zero if the job is inactive. SuperBASIC activates jobs at priority $20
jcb_wait *	$0014	word	job WAIT counter: >0 number of frame times to release
jcb.nsus	0		not suspended
jcb.wait	-1		wait forever
jcb.wjob	-2		wait for job
jcb_rela	$0016	byte	set if next IO call is RELative Address [QDOS, SMSQ]
jcb_prio	$0016	word	job priority (composite) [SMS2]
jcb_prab	$0016	byte	job priority (absolute) [SMS2]
jcb_prin	$0017	byte	job priority (increment) [SMS2]
jcb_wflg	$0017	byte	set if there is a job waiting on completion of this one [QDOS, SMSQ]
jcb_wjid	$0018	long	Waiting Job ID
jcb_exv	$001c	long	pointer to EXeption vector
jcb_save	$0020		job SAVE area
jcb_d0	$0020		saved d0
jcb_d1	$0024		saved d1
jcb_d2	$0028		saved d2
jcb_d3	$002c		saved d3
jcb_d4	$0030		saved d4
jcb_d5	$0034		saved d5
jcb_d6	$0038		saved d6
jcb_d7	$003c		saved d7
jcb_a0	$0040		saved a0
jcb_a1	$0044		saved a1
jcb_a2	$0048		saved a2
jcb_a3	$004c		saved a3
jcb_a4	$0050		saved a4
jcb_a5	$0054		saved a5
jcb_a6	$0058		saved a6
jcb_a7	$005c		saved a7
jcb_sr	$0060		saved sr
jcb_ccr	$0061		saved ccr
jcb_pc	$0062		saved pc
jcb_reln	$0066	byte	set if next IO call is RELative Address [SMS2]
jcb_evts	$0066	byte	8 bit event vector [SMSQ 2.71+]
jcb_evtw	$0067	byte	8 bit events waited for [SMSQ 2.71+]
jcb_end	$0068		end of header

Thus the job identified by job-ID starts at ((**SYS_JBTB**)+4*job-ID.w), and the most significant word of job-ID must match the tag held at **JCB_TAG** on from this address (otherwise that job no longer exists). A negative job-ID implies that the job no longer exists, as does a value of job-ID.w which is greater than the length of the job table held in **SYS_JBTP**.

Entries marked by * should not be modified. Other entries may be modified by a trap, or may be changed directly with caution.

18.6 Memory Block Table Definitions

The following keys define the format of a slave block table entry.

sbt_stat	$00	byte	STATus of block - see below
sbt_phys	$01	byte	PHYsical sector on drive [DD2]
sbt_prio	$01	byte	block priority [QL]
sbt_phyg	$02	word	PHYsical group on drive [DD2]
sbt_sect	$02	word	sector number (Microdrive*2) [QL]
sbt_file	$04	word	FILE number
sbt_blok	$06	word	BLOcK number
sbt_end	$08		

sbt.len	$0008	length of slave block table entry
sbt.size	$0200	size of slave block

The most significant 4 bits of the status byte contain the pointer to the physical device block **SYS_FSDD**, the least significant are the status codes:
status byte usage

sbt.unav	%0000	block is unavailable to the file system
sbt.mpty	%0001	block eMPTY
sbt.read	%1001	awaiting READ
sbt.true	%0011	block is TRUE representation of file
sbt.veri	%1011	awaiting VERIfy
sbt.writ	%0111	awaiting WRITe (updated)

Masks:

sbt.driv	%11110001+$ffffff00	mask of pointer to DRIVe
sbt.drvv	%11110011+$ffffff00	mask of DRiVe Valid bits
sbt.stat	%00001111	mask of STATus bits
sbt.actn	%00001100	mask of ACTioN bits
sbt.inus	%00001110	mask of IN USe bits

slave block status bits (least significant four)

sbt..fsb	0	Filing System Block
sbt..rrq	3	Read ReQuest
sbt..wrq	2	Write ReQuest
sbt..vld	1	block is VaLiD

18.7 Channel Definitions

The position of a channel definition block corresponding to a given channel ID can be found using a similar method to that used for finding the block for a job described in section 3.1. The relevant system variables are **SYS_CHTB** and **SYS_CHTT**.

Channel definition header for all channels:

chn_len	$0000	long	LENgth of channel block
chn_drvr	$0004	long	address of driver linkage
chn_ownr	$0008	long	OWNeR of this channel
chn_rflg	$000c	long	pointer to channel Closed FLaG in channel table, MSB set to $ff on close
chn_tag	$0010	word	channel TAG
chn_stat	$0012	byte	STATus 0 ok, $ff waiting (A1 abs), $80 waiting (A1 rel A6)
chn_actn	$0013	byte	IO action (stored value of d0)
chn_jbwt	$0014	long	JoB WaiTing for IO
chn_end	$0018		end of header

Extended channel definition for Pipes (plain serial queues):

chn_qin	$0018	long	pointer to input queue (or 0 if output pipe)
chn_qout	$001c	long	pointer to output queue (or 0 if input pipe)
chn_qend	$0020		end of definition (for input pipe) or queue header followed by queue (for output pipe)

Device driver header:

chn_next	$0000	long	pointer to next driver
chn_inot	$0004	long	entry for input and output
chn_open	$0008	long	entry for open
chn_clos	$000c	long	entry for close

The following are for directory devices (file system) only:

chn_slav	$0010	long	entry for slaving blocks
chn_renm	$0014	long	entry for rename [QL]
chn_frmt	$001c	long	entry for format medium
chn_dfln	$0020	long	length of physical definition block
chn_dnam	$0024	string	drive name

18.8 File System Definition Blocks:

chn_link	$0018	long	LINKed list of channel blocks
chn_accs	$001c	byte	ACCeSs mode
chn_drid	$001d	byte	DRive ID
chn_qdid	$001e	word	Qdos thinks this is file ID
chn_fpos	$0020	long	File POSition
chn_feof	$0024	long	File EOF
chn_csb	$0028	long	current slave block
chn_updt	$002c	byte	file UPDaTed
chn_usef	$002d	byte	file USE Flags [DD2]
chn..usd	7		file used
chn..dst	0		date set
chn..vst	1		version set
chn_name	$0032	string	file NAME
chn.nmln	$24		max file NaMe LeNgth
chn_ddef	$0058	long	pointer to physical definition block [DD2]
chn_drnr	$005c	word	DRive NumbeR [DD2]
chn_flid	$005e	word	FiLe ID [DD2]
chn_sctl	$005e	word	SeCTor Length (direct sector IO) 0:128 1:256 etc [DD2]
chn_opwk	$0060	long	$40 (hdr.len) bytes of working space for open [DD2]
chn_sdid	$0062	word	(Sub-)Directory ID [DD2]
chn_sdps	$0064	long	(Sub-)Directory entry PoSition [DD2]
chn_sdef	$0068	long	(Sub-)Directory End of File (wrong if IOA.KDIR) [DD2]
chn_spr	$0070	$30 b	spare [DD2]
chn_fend	$00a0		File system channel end [DD2]

The common part of a physical definition block

fs.nmlen	$24		max length of file name
fs.hdlen	$40		length of file system header
fs_drivr	$10	long	pointer to driver
fs_drivn	$14	byte	drive number
fs_mname	$16	string	medium name (maximum ten characters)
fs_files	$22	byte	number of files open

18.8.1 Microdrive Physical Definition Block [QL]

md_fail	$24	byte	failure count - this increases by 1 with every revolution for each operation until it either reaches 4 (for write or verify) or 8 (for read), after which the system notifies a file error.
md_spare	$25	3 bytes	
md_map	$28	$ff*2 b	microdrive sector map
md_lsect	$226	word	number of last sector allocated
md_pendg	$228	$100 w	map of pending operations - a word for each sector
md_end	$428		

18.9 Device Driver Linkage Block

for details refer to section 7.1

iod_sqfb	-$08	long	SMSQ IO facility bits
iod..ssr	0		bit set for serial
iod..swi	1		bit set for window operations
iod..sfi	2		bit set for filing system ops
iod..sdl	8		bit set for delete
iod..ssb	16		bit set for slave block
iod..scn	18		bit set for channel name
iod..sfm	19		bit set for format
iod..sdd	20		bit set for directory device
iod_sqio	-$04	long	SMSQ IO compatible flag
iod.sqio	'SQIO'		
iod_xilk	$00	long	external interrupt linkage
iod_xiad	$04	long	external interrupt service routine address
iod_pllk	$08	long	polling interrupt linkage
iod_plad	$0c	long	polling interrupt service routine address
iod_shlk	$10	long	scheduler loop linkage
iod_shad	$14	long	scheduler loop service routine address
iod_iolk	$18	long	io driver linkage
iod_ioad	$1c	long	input / output routine address
iod_open	$20	long	open routine address
iod_clos	$24	long	close routine address
iod_iend	$28		end of minimum device driver linkage
iod_fslv	$28	long	forced slaving address
iod_spr1	$2c		spare
iod_cnam	$30	long	set channel name
iod_frmt	$34	long	format routine address
iod_plen	$38	long	Physical definition block LENgth
iod_dnus	$3c	string	Drive Name (current USage)
iod_dnam	$42	string	Drive NAMe [SMSQ]

18.9.1 Screen Driver Data Block Definition

sd_xmin	$18	word	window top LHS
sd_ymin	$1a	word	
sd_xsize	$1c	word	window size
sd_ysize	$1e	word	
sd_borwd	$20	word	border width
sd_xpos	$22	word	cursor position
sd_ypos	$24	word	
sd_xinc	$26	word	cursor increment
sd_yinc	$28	word	
sd_font	$2a	2*long	font addresses
sd_scrb	$32	long	base address of screen
sd_pmask	$36	long	paper colour mask
sd_smask	$3a	long	strip colour mask
sd_imask	$3e	long	ink colour mask

sd_cattr	$42	byte	character attributes
sd..unot	0		underline mode
sd..flsh	1		flash mode
sd..strp	2		transparent strip
sd..xor	3		XOR mode
sd..hi	4		double height characters
sd..wide	5		extended width characters
sd..dbl	6		double width characters
sd..grf	7		graphics positioned character
sd_curf	$43	byte	cursor flag 0=suppressed, >0=visible, <0 invisible
sd_pcolr	$44	byte	paper colour byte
sd_scolr	$45	byte	strip colour byte
sd_icolr	$46	byte	ink colour byte
sd_bcolr	$47	byte	border colour byte
sd_nlsta	$48	byte	new line status (>0 implicit, <0 explicit) [SMS]
sd_nlsta	$48	byte	new line status (>0 pending, <0 done). [QDOS]
sd_fmod	$49	byte	fill mode (0=off, 1=on)
sd_yorg	$4a	float	graphics window y-origin
sd_xorg	$50	float	graphics window x-origin
sd_scal	$56	float	graphics scale factor
sd_fbuf	$5c	long	pointer to fill buffer
sd_fuse	$60	long	pointer to user-defined fill vectors [QL]
sd_linel	$64	word	line length in bytes [QDOS V1.10+]
sd_end	$68		length of screen driver [QODS V1.10+]
sd_end	$66		... in QDOS before V1.10

18.9.2 Serial channel Definition Block [QL]

ser_chnq	$18	word	port number: 1 or 2
ser_par	$1a	word	parity: 0 none, 1 odd, 2 even, 3 mark, 4 space
ser_thxs	$1c	word	transmit handshake flag: -1 ignore, 0 handshake
ser_prot	$1e	word	protocol flag: -1 for R, 0 for Z, +1 for C
ser_rxq	$20	$62 b	receive queue header followed by queue
ser_txq	$82	$62 b	transmit queue header followed by queue
ser_end	$e4		

18.9.3 Network channel Definition Block [QL]

net_hedr	$18	byte	destination station number
net_self	$19	byte	number of station which opened channel
net_blkl	$1a	byte	lsb of data block number
net_blkh	$1b	byte	msb of data block number
net_type	$1c	byte	packet type: 0 for data, 1 last packet (EOF)
net_nbyt	$1d	byte	number of bytes in data block
net_dchk	$1e	byte	data checksum
net_hchk	$1f	byte	header checksum
net_data	$20	$ff b	data block
net_rpnt	$11f	byte	pointer to current position in data block
net_end	$120		

18.10 Queue Header Definitions

The following is the format of the header of a queue manipulated using the system's built-in queue handling routines.

q_eoff	$00	byte	end of file flag (MSbit)
q_nextq	$00	long	link to next queue
q_end	$04	long	pointer to end of queue
q_nextin	$08	long	pointer to next location to put byte in
q_nxtout	$0c	long	pointer to next location to take byte from
q_queue	$10		start of queue

18.11 Arithmetical Interpreter Operation Codes

The following are the codes for the operations which can be performed on the QL through the vectored routines which access the arithmetic interpreter.

qa.end	$00	END of multiple operation
qa.nint	$02	round fp to Nearest INTeger
qa.int	$04	truncate fp to INTeger
qa.nlint	$06	round fp to Nearest Long INTeger
qa.float	$08	FLOAT integer
qa.add	$0a	ADD (top of stack to next of stack)
qa.sub	$0c	SUBtract (tos from nos)
qa.mul	$0e	MULtiply (tos by nos)
qa.div	$10	DIVide (tos into nos)
qa.abs	$12	ABSolute value
qa.neg	$14	NEGate
qa.dup	$16	DUPlicate
qa.cos	$18	COSine
qa.sin	$1a	SINe
qa.tan	$1c	TANgent
qa.cot	$1e	COTangent
qa.asin	$20	ArcSINe
qa.acos	$22	ArcCOSine
qa.atan	$24	ArcTANgent
qa.acot	$26	ArcCOTangent
qa.sqrt	$28	SQuare RooT
qa.log	$2a	Log (Natural)
qa.l10	$2c	Log base 10
qa.exp	$2e	Exponential
qa.pwrf	$30	raise to PoWeR (Floating point) (nos to power of tos)
qa.maxop	$30	

The following arithmetic-keys are available only in SMS2, SMSQ and Minerva:

qa.one	$01	push constant 1 (float)
qa.zero	$03	push constant 0 (float)
qa.n	$05	followed by a signed byte, to push -128 to 127 (float)
qa.k	$07	plus a byte, nibbles select mantissa and adjust exponent. Following byte values may be:
		qak.pi180 $56
		qak.loge $69
		qak.pi6 $79
		qak.ln2 $88-$100
		qak.sqrt3 $98-$100
		qak.pi $a8-$100
		qak.pi2 $a7-$100
qa.flong	$09	float a long integer
qa.halve	$0d	TOS / 2
qa.doubl	$0f	TOS * 2
qa.recip	$11	1 / TOS
qa.roll	$13	(TOS)B, C, A -> (TOS)A, B, C (roll 3rd to top)
qa.over	$15	adjust stack, NOS-> TOS
qa.swap	$17	NOS <-> TOS
qa.arg	$25	arg(TOS,NOS)=a, solves TOS=k*cos(a) & NOS=k*sin(a)
qa.mod	$27	sqrt(TOS^2+NOS^2)
qa.squar	$29	TOS * TOS
qa.power	$2f	NOS ^ TOS, where TOS is a signed short int

qa.load	$00	keys for load and store
qa.stor	$01	

18.12 IPC Link Commands

These can be used with the **SMS.HDOP** trap.

rset_cmd	0	system reset [QL]
stat_cmd	1	report input status [QL]
ops1_cmd	2	open RS232 channel 1 [QL]
ops2_cmd	3	open RS232 channel 2 [QL]
cls1_cmd	4	close RS232 channel 1 [QL]
cls2_cmd	5	close RS232 channel 2 [QL]
rds1_cmd	6	read RS232 channel 1 [QL]
rds2_cmd	7	read RS232 channel 2 [QL]
rdkb_cmd	8	read keyboard [QL]
kbdr_cmd	9	keyboard read directly
inso_cmd	10	initiate sound process
kiso_cmd	11	kill sound process
mdrs_cmd	12	microdrive reduced sensitivity [QL]
baud_cmd	13	change baud rate [QL]
rand_cmd	14	random number generator [QL]
test_cmd	15	test [QL]

18.13 Hardware Keys

The following are the addresses of the registers within the QL hardware. [QL]

pc_clock	$18000	real time clock in seconds (long word)

The following are the masks used to access the transmit control register (pc_tctrl and sys_tmod).

pc_tctrl	$18002	transmit control
pc..sern	3	serial port number or 0=mdv, 1=net
pc..serb	4	0=serial IO, 1=mdv or net
pc..diro	7	direct output bit
pc.bmask	%00000111	system baud rate
pc.notmd	%11100111	all bits except mode control
pc.mdvmd	%00010000	microdrive mode (set if you can access microdrives)
pc.netmd	%00011000	network mode (set if you can access net)

pc_ipcwr	$18003	IPC write
pc.ipcwr	%00001100	IPC write bit
pc..ipcw	1	... 1
pc.ipcrd	%00001110	IPC read bit

The following is the format of the microdrive control/systems register.

pc_mctrl	$18020	microdrive control status and IPC status

If you write to this register, the following bits can be used:

pc..sel	0	mcirodrive select bit
pc..sclk	1	microdrive select clock bit
pc..writ	2	microdrive write (set=enable write)
pc..eras	3	microdrive erase (set=enable erase)

The following masks can therefore be useful:

pc.read	%0010	read (or idle) mode
pc.select	%0011	select bit set
pc.desel	%0010	select bit not set
pc.erase	%1010	enable erase/stop write to drive
pc.write	%1110	enable both erase and write to drive

If you rad the register, you will however, have access to the following information in the specified bits:

pc_ipcrd	$18020	IPC read (is the same)
pc..txfl	1	set if microdrive Xmit buffer is full
pc..rxrd	2	set if microdrive read buffer is ready
pc..gap	3	gap
pc..dtr1	4	DTR on port 1 (clear if device is ready)
pc..cts2	5	CTS on port 2 (clear if device is ready)
pc..ipca	6	IPC acknowledge
pc..ipcd	7	IPC data bit

The following is the format of the interrupt register.

pc_intr	$18021	interrupt control/status
pc.intrg	%00000001	gap interrupt
pc.intri	%00000010	interface interrupt
pc.intrt	%00000100	transmit interrupt
pc.intrf	%00001000	frame interrupt
pc.intre	%00010000	external interrrupt
pc.maskg	%00100000	gap mask
pc.maski	%01000000	interface mask
pc.maskt	%10000000	transmit mask

pc_tdata	$18022	transmit data
pc_trak1	$18022	microdrive read track 1
pc_trak2	$18023	microdrive read track 2

The following ist the format of the display control register.

mc_stat	$18063	display control register
mc..blnk	1	blanks display
mc..m256	3	sets MODE 8 (256 pixels across)
mc..scrn	7	sets the screen base ($20000 or $28000, if set)

The following is a list of addresses available when a QIMI (QL Internal Mouse Interface) is installed in a QL. Warning: you should not access the mouse via these hardware addresses, you should always access it by using the Pointer Interface!

mi_button	$1bf9c	Mouse button state
mib..left	4	left button
mib..rigth	5	right button
mi_status	$1bfbc	Status register
mis..diry	0	Y direction
mis..intx	2	Interrupt X direction
mis..dirx	4	X direction
mis..inty	5	Interrupt Y direction
mi_clrint	$1bfbe	Clear interrupt service

18.14 Trap Keys

This section gives a summary of all of the Qdos traps, together with their access keys passed in D0. All keys are in hex.

18.14.1 Trap 1 Keys (System Traps)

do.sms2	1	SMS2 trap entry
do.smsq	1	SMSQ trap entry
sms.myjb	-1	SMS key for MY JoB
sms.info	$00	get INFOrmation on SMS
sms.crjb	$01	CReate JoB
sms.injb	$02	get INformation on JoB
sms.rmjb	$04	ReMove JoB
sms.frjb	$05	Forced Remove JoB
sms.frtp	$06	find largest FRee space in TPa
sms.exv	$07	set EXception Vector
sms.ssjb	$08	SuSpend a JoB
sms.usjb	$09	UnSuspend a JoB
sms.acjb	$0a	ACtivate a JoB
sms.spjb	$0b	Set Priority of JoB
sms.alhp	$0c	ALlocate in HeaP
sms.rehp	$0d	RElease to HeaP
sms.arpa	$0e	Allocate in Resident Procedure Area
sms.dmod	$10	set or read the Display MODe
sms.hdop	$11	do a Hardware Dependent OPeration
sms.comm	$12	set COMMuncation baud rate etc.
sms.rrtc	$13	Read Real Time Clock
sms.srtc	$14	Set Real Time Clock
sms.artc	$15	Adjust Real Time Clock
sms.ampa	$16	Allocate space in Moveable Program Area (SuperBASIC)
sms.rmpa	$17	Release space to Moveable Program Area (SuperBASIC)
sms.achp	$18	Allocate space in Common HeaP
sms.rchp	$19	Release space in Common HeaP
sms.lexi	$1a	Link in EXternal Interrupt action
sms.rexi	$1b	Remove EXternal Interrupt action
sms.lpol	$1c	Link in POLled action
sms.rpol	$1d	Remove POLled action
sms.lshd	$1e	Link in ScHeDuler action
sms.rshd	$1f	Remove ScHeDuler action
sms.liod	$20	Link in IO Device driver
sms.riod	$21	Remove IO Device driver
sms.lfsd	$22	Link in Filing System Device driver
sms.rfsd	$23	Remove Filing System Device driver
sms.trns	$24	Set (QDOS) TRaNSlate or messages [QDOS V1.10+]
sms.xtop	$25	eXTernal OPeration [SMSQ]

sms.lthg	$26	Link in THinG [SMS, EXT]
sms.rthg	$27	Remove THinG [SMS, EXT]
sms.uthg	$28	Use THinG [SMS, EXT]
sms.fthg	$29	Free THinG [SMS, EXT]
sms.zthg	$2a	Zap THinG [SMS, EXT]
sms.nthg	$2b	Next THinG [SMS, EXT]
sms.nthu	$2c	Next Thing User [SMS, EXT]
sms.iopr	$2e	IO PRiority [SMSQ]
sms.cach	$2f	CACHe handling [SMSQ]
sms.lldm	$30	Link in Language Dependent Module(s) [SMSQ]
sms.lenq	$31	Language ENQuiry [SMSQ]
sms.lset	$32	Language SET [SMSQ]
sms.pset	$33	Printer translate SET [SMSQ]
sms.mptr	$34	find a Message PoinTeR [SMSQ]
sms.fprm	$35	Find PReferred Module [SMSQ]
sms.schp	$38	Shrink alloaction in common heap [SMSQ]
sms.sevt	$3a	Send event to job [SMSQ 2.71+]
sms.wevt	$3b	Wait for event [SMSQ 2.71+]

18.14.2 Trap 2 Keys (I/O Allocation Traps)

do.ioa	2	trap #2
do.rlioa	4	trap #4
ioa.open	$01	OPEN IOSS channel
ioa.clos	$02	CLOSe IOSS channel
ioa.frmt	$03	FoRMaT medium on device
ioa.delf	$04	DELete file from device
ioa.sown	$05	Set OWNer of channel
ioa.cnam	$06	Fetch channel name

Ownership keys

no.owner	0
myself	-1

IOA.OPEN keys (d3.b)

ioa.kexc	$00	Key for EXClusive use (read/write)
ioa.kshr	$01	Key for SHaRed access (read only)
ioa.knew	$02	Key for NEW file (empty, read/write)
ioa.kovr	$03	Key for OVeRwrite (delete contents if it exists)
ioa.kdir	$04	Key for DIRectory file
ioa.krnm	$05	Key for ReNaMe [DD2]

18.14.3 Trap 3 Keys (I/O Traps)

do.io	3	trap #3
do.relio	4	trap #4
iob.test	$00	TEST input
iob.fbyt	$01	Fetch BYTe from input
iob.flin	$02	Fetch LINe from input
iob.fmul	$03	Fetch MULtiple characters/bytes
iob.elin	$04	Edit LINe of characters
iob.sbyt	$05	Send BYTe to output
iob.smul	$07	Send MULtiple bytes
iow.xtop	$09	eXTernal OPeration on screen
iow.pixq	$0a	PIXel coordinate Query
iow.chrq	$0b	CHaRacter coordinate Query
iow.defb	$0c	DEFine Border
iow.defw	$0d	DEFine Window
iow.ecur	$0e	Enable CURsor
iow.dcur	$0f	Disable CURsor
iow.scur	$10	Set CURsor position (character coordinates)
iow.scol	$11	Set cursor COLumn
iow.newl	$12	put cursor on a NEW Line
iow.pcol	$13	move cursor to Previous COLumn
iow.ncol	$14	move cursor to Next COLumn
iow.prow	$15	move cursor to Prevous ROW
iow.nrow	$16	move cursor to Next ROW
iow.spix	$17	Set cursor to PIXel position
iow.scra	$18	SCRoll All of window
iow.scrt	$19	SCRoll Top of window (above cursor)
iow.scrb	$1a	SCRoll Bottom of window (below cursor)
iow.pana	$1b	PAN All of window
iow.panl	$1e	PAN cursor Line
iow.panr	$1f	PAN Right hand end of cursor line
iow.clra	$20	CLeaR All of window
iow.clrt	$21	CLeaR Top of window (above cursor)
iow.clrb	$22	CLeaR Bottom of window (below cursor)
iow.clrl	$23	CLeaR cursor Line
iow.clrr	$24	CLeaR Right hand side of cursor line
iow.font	$25	set / read FOuNT (font U.S.A.)
iow.rclr	$26	ReCoLouR a window
iow.spap	$27	Set PAPer colour
iow.sstr	$28	Set STRip colour
iow.sink	$29	Set INK colour
iow.sfla	$2a	Set FLash Attribute
iow.sula	$2b	Set UnderLine Attribute
iow.sova	$2c	Set OVerwrite Attributes
iow.ssiz	$2d	Set character SIZe
iow.blok	$2e	fill a BLOcK with colour
iow.donl	$2f	DO a pending newline
iog.dot	$30	draw (list of) DOTs
iog.line	$31	draw (list of) LINEs
iog.arc	$32	draw (list of) ARCs
iog.elip	$33	draw ELlIPse
iog.scal	$34	set graphics SCALe
iog.fill	$35	set area FILL
iog.sgcr	$36	Set Graphics CuRsor position

iof.chek	$40	CHEcK all pending operations on file
iof.flsh	$41	FLuSH all buffers
iof.posa	$42	set file POSition to Absolute address
iofp.off	$F0FFF0FF	key in d1 returns sector 0 offset (direct sector access)
iof.posr	$43	move file POSition Relative to current position
iof.minf	$45	get Medium INFormation
iof.shdr	$46	Set file HeaDeR
iof.rhdr	$47	Read file HeaDeR
iof.load	$48	(scatter) LOAD file
iof.save	$49	(scatter) SAVE file
iof.rnam	$4a	ReNAMe file [EXT]
iof.trnc	$4b	TRuNCate file to current position [EXT]
iof.date	$4c	set or get file DATEs [EXT,DD2]
iofd.get	-1	d1 key, GET date (or version)
iofd.cur	0	d1 key, set CURrent date (or version)
iofd.upd	0	d2 key, set/get UPDate date
iofd.bak	2	d2 key, set/get BAcKup date
iof.mkdr	$4d	MaKe DiRectory [DD2]
iof.vers	$4e	set or get VERSion (d1 keys as iof.date) [DD2]
iof.xinf	$4f	get eXtended INFormation [DD2]

All keys higher than $4f are for pointer-driven CON devices. Please refer to the QPTR manual.

Timeout keys

no.wait	0
forever	-1

18.15 List of Vectored Routines

The following is a list of the vectored routines, together with the addresses of their associated vectors.

mem.achp	$00c0	Allocate space in Common HeaP
mem.rchp	$00c2	Return space to Common HeaP
mem.alhp	$00d8	ALlocate in HeaP
mem.rehp	$00da	REturn to HeaP
mem.llst	$00d2	Link into LiST
mem.rlst	$00d4	Remove from LiST
opw.wind	$00c4	Open WINDow using name
opw.con	$00c6	Open CONsole
opw.scr	$00c8	Open SCReen
ut.wersy	$00ca	Write an ERror to SYstem window
ut.werms	$00cc	Write an ERror MeSsage
ut.wint	$00ce	Write an INTeger
ut.wtext	$00d0	Write TEXT
ut.cstr	$00e6	Compare STRings
ioq.setq	$00dc	SET up a Queue in standard form
ioq.test	$00de	TEST a queue for pending byte / space available
ioq.pbyt	$00e0	Put a BYTe into a queue
ioq.gbyt	$00e2	Get a BYTe out of a queue
ioq.seof	$00e4	Set EOF in queue

iou.ssq	$00e8	Standard Serial Queue handling
iou.ssio	$00ea	Standard Serial IO
iou.dnam	$0122	decode Device NAMe
cv.datil	$00d6	DATE and time (6 words) to Integer Long [SMS]
cv.ildat	$00ec	Integer (Long) to DAte and Time string
cv.ilday	$00ee	Integer (Long) to DAY string
cv.fpdec	$00f0	Floating Point to ascii DECimal
cv.iwdec	$00f2	integer (word) to ascii decimal
cv.ibbin	$00f4	integer (byte) to ascii binary
cv.iwbin	$00f6	integer (word) to ascii binary
cv.ilbin	$00f8	integer (long) to ascii binary
cv.ibhex	$00fa	integer (byte) to ascii hexadecimal
cv.iwhex	$00fc	integer (word) to ascii hexadecimal
cv.ilhex	$00fe	integer (long) to ascii hexadecimal
cv.decfp	$0100	decimal to floating point
cv.deciw	$0102	decimal to integer word
cv.binib	$0104	binary ascii to integer (byte)
cv.biniw	$0106	binary ascii to integer (word)
cv.binil	$0108	binary ascii to integer (long)
cv.hexib	$010a	hexadecimal ascii to integer (byte)
cv.hexiw	$010c	hexadecimal ascii to integer (word)
cv.hexil	$010e	hexadecimal ascii to integer (long)
sb.inipr	$0110	INITialise PRocedure table
sb.gtint	$0112	GeT INTeger
sb.gtfp	$0114	GeT Floating Point
sb.gtstr	$0116	GeT STRing
sb.gtlin	$0118	GeT Long INteger
sb.putp	$0120	PUT Parameter
qa.resri	$011a	QL Arithmetic Reserve Room on stack
qa.op	$011c	QL Arithmetic OPeration
qa.mop	$011e	QL Arithmetic Multiple OPeration

From now on add $4000 to all.

md.read	$0124	Microdrive: read a sector [QL]
md.write	$0126	Microdrive: write a sector [QL]
md.verif	$0128	Microdrive: verify a sector [QL]
md.rdhdr	$012a	Microdrive: read a sector header [QL]
sb.parse	$012c	parse; (a2) points to table
sb.graph	$012e	main syntax graph
sb.expgr	$0130	expression graph
sb.strip	$0132	strip spaces from tokenised line
sb.paerr	$0134	parser error
sb.ledit	$0136	edit line into program (just line number deletes)
sb.expnd	$0138	expand / print line(s) (+$4004 A4 points to program)
sb.paini	$013a	initialise parser

18.16 Keys for Things

The following are keys for the Thing linkage block. The items marked with * are filled in by LTHG.

th_nxtth *	$00	long	link to NeXT THing
th_usage *	$04	long	thing's USAGE list
th_frfre *	$08	long	address of "close" routine for FoRced FREe
th_frzap *	$0c	long	address of "close" routine for FoRced ZAP
th_thing	$10	long	pointer to THING itself
th_use	$14	long	code to USE a thing
th_free	$18	long	code to FREE a thing
th_ffree	$1c	long	code to Force FREE a thing
th_remov	$20	long	code to tidy before REMOVing thing
th_nshar	$24	byte	Non-SHAReable Thing if top bit set
th_check *	$25	byte	CHECK byte
th_verid	$26	long	version ID
th_name	$2a	string	name of thing
th.len	$2c		basic length of thing linkage

Usage list header/entry

thu_link	$10	long	link to first/next usage block
thu.ulnk	$20		size of usage list header/entry

Standard Thing header (offsets are relative to thh_flag)

thh_flag	$00	long	Thing header flag
thh.flag	'THG%'		standard value of thing header flag
thh_type	$04	long	type of thing
tht..lst	24		bit set for list of things
tht.util	$00000000		utility thing
tht.exec	$00000001		executable thing
tht.data	$00000002		shared data
tht.extn	$01000003		extensions (user mode)
tht.exts	$01000004		extensions for system (supervisor mode)

Thing Itself Header (after Standard Thing Header)

thh_entr	$08		Thing ENTRy routine
thh_exec	$0c		Thing EXEC routine

List of Things header (after Standard Thing Header)

thh_next	$08	long	offset of next (or 0)
thh_exid	$0c	long	extra ID

Executable Thing header extension (after Standard Thing Header)

thh_hdrs	$08	long	offset of start of header
thh_hdrl	$0c	long	length of header
thh_data	$10	long	size of data area rired
thh_strt	$14	long	offset of start of code (0 to start at header)

Extension Thing Header (after Standard Thing Header and List of Things Header)

thh_pdef	$10	long	offset of parameter definitions or 0
thh_pdes	$14	long	offset of parameter descriptions or 0
thh_code	$18		start of code

Thing parameter definitions

thp.rep	$ffff	start and end delimiter for repeated group

thp..ptr	15	bit set for pointer parameter
thp..cal	14	bit set for call parameter
thp..ret	13	bit set for return paramter
thp..opt	12	bit set if parameter is optional
thp..nnl	11	bit set if negative for null - NOT thp..ptr
thp..arr	11	bit set for array - thp..ptr

thp..sgn	1	bit set if value is signed
thp..chr	2	bit set if character allowed
thp..byt	3	bit set if byte value allowed/rired
thp..wrd	4	bit set if word value allowed/rired
thp..lng	5	bit set if long value allowed/rired
thp..cid	6	bit set for channel ID
thp..fp8	7	bit set for eight byte floating point

The following bits are only allowed for pointer parameters:

thp..str	8	standard string
thp..sst	9	sub-string

thp.char	$0004	character
thp.ubyt	$0008	unsigned byte
thp.sbyt	$000a	signed byte
thp.uwrd	$0010	unsigned word
thp.swrd	$0012	signed word
thp.ulng	$0020	unsigned long
thp.slng	$0022	signed long
thp.chid	$0040	channel ID
thp.fp8	$0082	eight byte floating point
thp.str	$0100	string
thp.sstr	$0200	sub-string
thp.nnul	1<<thp..nnl	negative null (-1)
thp.arr	1<<thp..arr	array
thp.opt	1<<thp..opt	optional
thp.upd	1<<thp..ptr+1<<thp..cal+1<<thp..ret	updated parameter
thp.call	1<<thp..ptr+1<<thp..cal	call parameter
thp.ret	1<<thp..ptr+1<<thp..ret	return parameter
thp.ptr	1<<thp..ptr	call or return parameter

18.17 Keys for HOTKEY Thing

HOTKEY linkage block:

hk.fitem	$0014	find item
hk.crjob	$0018	hotkey create job
hk.kjob	$001c	hotkey kill job
hk.set	$0020	hotkey set
hks.off	-1	turn off
hks.on	0	turn on
hks.rset	1	reset
hks.set	2	set
hk.remov	$0024	hotkey remove
hk.do	$0028	hotkey do
hk.stbuf	$002c	hotkey stuff buffer
hk.gtbuf	$0030	hotkey get buffer (d0=0 current -1 prev)
hk.guard	$0034	hotkey guardian / grabber (V2.04 onwards)

The HOTKEY item:

hki_id	$0000	word	hotkey id
hki.id	'hi'		
hki_type	$0002	word	hotkey item type
hki..trn	0		bit set if item is transient thing
hki.llrc	-8		last line recall
hki.stpr	-6		stuff kbd with prevous string from buffer
hki.stbf	-4		stuff keyboard queue from buffer
hki.stuf	-2		stuff keyboard queue with string
hki.cmd	0		pick SuperBASIC and stuff command
hki.nop	2		just do code
hki.xthg	4		execute thing
hki.xttr	5		as hki.xthg but thing is transient
hki.xfil	6		execute file
hki.pick	8		pick job
hki.wake	10		pick and wake job (execute thing if fails)
hki.wktr	11		as hki.wake but thing is transient
hki.wkxf	12		pick and wake job (execute file if fails)
hki_ptr	$0004	long	pointer to (preprocessing) code, stuff buffer
hki_name	$0008	string	item name

Executable program header definitions:

hkh.hlen	10	header length for zero length name
hkh.plen	20	preamble length

hkh_jsgd	$00	JSR [$4eb9]
hkh_gard	$02	... guardian
hkh_wdef	$06	window definition
hkh.unlk	-1	guardian window size for unlockable
hkh.nogd	0	guardian window size for no guardian
hkh_brdr	$0e	border colour
hkh_gmem	$10	memory (in KBytes)
hkh_jpa6	$12	JMP (A6) [$4ed6]

19. SMSQ

This chapter deals specifically with SMSQ (and SMSQ/E, of course). It is a separate chapter so that you can see the advantages of SMSQ at one glance. All the descriptions listed here will be referenced from the other chapters later, and additional traps will also be put into the right chapters 13 to 15. Some features are integrated into the relevant parts of the manual already.

As SMSQ/E is a growing system which will be expanded depending on user's requirements, this manual can reflect the features of SMSQ at the current situation only. It is quite possible that a number of features are not available on earlier versions of SMSQ. At the time of writing, the version of SMSQ is V2.61. In case features are not supported by earlier versions, there should be no serious problem: unused system variables were set to 0, non-existing traps will either return ERR.IPAR or ERR.NIMP, or the call will have no effect at all.

19.1 Language handling in SMSQ

19.1.1 Principles

During normal operation, the "language" dependent parts of the operating system are maintained as tables appropriate to the "current" language. In order to ensure that current language may be changed, the system also maintains a list of language dependent modules. When the current language is changed, the list is scanned to find the appropriate language modules to be made current.

The language dependent module list, and the modules themselves, may be maintained in the filing system or in memory. The module structure is the same in either case.

19.1.2 Classification of Language Dependent Modules

The language dependent modules are classified according to their contents rather than their usage.

Printer Translate Tables
An "old format" printer translate table has a "table of tables" which is the language code (word) and two word pointers (relative to the address of the language code) to two translate tables. The first translate table has 256 bytes of direct single byte translates. The second translate table has a byte entry count followed by 4 byte entries terminated by a zero byte. For each non-zero character, if the first translate table entry is zero, then the second table is searched. The first byte of each four byte entry is the untranslated character, followed by the three bytes this character should be translated to.

Keyboard Tables
A keyboard table has a table of tables which is the language code (word) and two word pointers (relative to the address of the language code) to two keyboard tables. The first keyboard table is four sets characters generated by each key for the four combinations of the "shift" and "control" keys. The second is a table of "non-spacing idents" (^, ~ etc.) which is normally 256 bytes of zero. The form of these keyboard tables depends on the type of keyboard and the associated driver.

Message Tables
A message table is the language code (word) followed by a table of word pointers (relative to the address of the language code) to error or other messages. Messages are numbered from 1. The message codes are formed by combining the message number and the message group (shifted) and negating the result to form a code. The offset of a message pointer from the language code is twice the message number.

To provide compatibility with older formats, the first message (number = -1) follows directly after the table. This means that the first word in the table also defines the size of the table.

The system can have several message tables: the message codes are grouped. At present, there is a limit of 256 message groups (numbered from 0 to 1020 in steps of 4) with a maximum of 128 messages per group.

In order to find the "correct" message, a message code is split into a message group and offset.

```
        neg.w    d0                     make the code positive
        moveq    #$7f,d1
        and.w    d0,d1                  bits 0 to 6 are the message number
        sub.w    d1,d0                  bits 7 to 14 are the message group
        add.w    d1,d1                  shift to get offset in message table
        lsr.w    #5,d0                  shift to get group number
or
        add.w    d0,d0                  double up code
        neg.w    d0                     and make positive
        moveq    #0,d1
        move.b   d0,d1                  offset in message table
        clr.b    d0                     clear message number from group
        lsr.w    #6,d0                  and shift to get group number
```

Language Preference Tables

A language preference table defines the preferred default languages to be used if the required language modules cannot be found.

ldp_ireg	$00	4 chars	international car registration code, space filled
ldp_defs	$04	n words	table of preferred language codes, terminated by 0

The international car registration code makes it possible to specify the language, for example, as "D" for German.

In general, the first preferred language code in the table will be the same as the language code in the module linkage structure.

The default of last resort is the first language preference table in the language dependent module list.

19.1.3 Language Dependent Module Structure

There is a common structure which is used as a link for all the types of module. The first word of this structure is only used when linking in new language dependent modules. It allows several modules to be defined in one block and for them all to be linked in at the same time.

ldm_type	$00	word	type of module 0 = preference table 1 = keyboard table 2 = printer translate table 3 = message table
ldm_group	$02	word	module group e.g. for messages table modules, the message group.
ldm_lang	$04	word	language code - usually the international dialing code of the country of origin
ldm_next	$06	word	relative pointer to next module in this block, 0 for the last module in the block
ldm_module	$08	long	relative pointer to the module itself

19.1.4 Language Specification

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

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

19.1.5 Implementation

The initial implementation is memory resident and uses a table of pointers to the language dependent modules rather than a true list. Each of the pointers points to a language dependent module. If the table overflows, it is re-allocated.

In general, new language dependent modules are add to the end of the list, thus ensuring that the first language variation for each module that is linked in is the default default.

All the language preference tables are, however, placed at the start of the list: not only is the appropriate language preference table always available before the list is scanned, but also the system "default of defaults" is replaced by any user preferences added to the list.

System Variables

sys_xact	$0144	byte	set if printer translate is active
sys_xtab	$0146	long	pointer to printer translate tables
sys_erms	$014a	long	(QDOS compatible) pointer to message group 0
sys_mstab	$014e	long	pointer to a 256 long word table of pointers to message groups. All undefined message groups have a zero ptr.
sys_lang	$0166	word	current language code
sys_ldmlst	$0168	long	pointer to language dependent module list

19.1.6 SMSQ OS Entries

There are a number of SMSQ OS entries for handling language dependencies.

sms.trns	$24	QDOS compatible (MT.TRA) entry
sms.lldm	$30	link in language dependent modules
sms.lenq	$31	enquire language code
sms.lset	$32	set current language
sms.pset	$33	set printer translate tables
sms.mptr	$34	find message pointer
sms.fprm	$35	find preferred module

Trap #1 D0=$24 **SMS.TRNS**

QDOS compatible translate

Call parameters		Return parameters	
D1.L	printer translate code	D1	???
D2.L	message table address or 0	D2	preserved
		D3+	all preserved

Error returns:

IPAR D2 is odd or does not point to $4AFB flag

If D2 is not zero and it points to a message table with language code $4AFB, this address is used for message group 0.

The printer translate tables are then set according to the value in D1 (see sms.pset).

Trap #1 D0=$30 **SMS.LLDM**

Link in Language Dependent Module

Call parameters		Return parameters	
A1	pointer to language dependent module	A1	preserved

Error returns:

always okay

This links all the language dependent modules in the list (A1) into the language dependent module list.

SMS.LENQ

Trap #1 D0=$31

Language Enquiry

Call parameters		Return parameters	
D1.L	language code or 0	D1	language code
D2.L	car registration (space filled) or 0	D2	car registration
		D3+	all preserved

Error returns:

always okay

This finds the car registration code corresponding to the the language code in D1 (if not zero) or the language code corresponding to the international car registration letters (in the most significant bytes of D2, space filled) or, if both D1 and D2 are 0, the current language and car registration letters.

The current language code is not changed.

If no corresponding language code can be found, the default language (the first language preference linked in by sms.lldm) is returned.

SMS.LSET

Trap #1 D0=$32

Language Set

Call parameters		Return parameters	
D1.L	language code or 0	D1	language code
D2.L	car registration (space filled) or 0	D2	car registration
		D3+	all preserved

Error returns:

always okay

This finds the car registration code corresponding to the the language code in D1 (if not zero) or the language code corresponding to the international car registration letters (in the most significant bytes of D2, space filled) or, if both D1 and D2 are 0, the current language and car registration letters.

The current language code is set to the returned value of D1.

If no corresponding language code can be found, the default language (the first language preference linked in by sms.lldm) is set.

SMS.PSET

Trap #1 D0=$33

Set Printer Translate

Call parameters		Return parameters	
D1.L	printer translate code	D1	???

Error returns:

always okay

This sets the printer translate tables according to the value in D1.

There are three printer translate codes which provide backwards compatibility with the QDOS MT.TRA call.
- To disable translate, D1 should be 0.
- To (re-)enable translate, D1 should be 1.
- To set a user translate, D1 should be the address of a special translate table (language code $4AFB).

With D1 = 1, the operation is not fully QDOS compatible in that, if a user translate has been requested, then the call to (re-)enable the translate will retain the user translate address. This is a facility which was not available in QDOS.

There are two new codes to set a language dependent table and two to set language independent translates.
- To select a language dependent translate without enabling the translate, the language code should be in the MSW of D1 and the LSW should be -1.
- To select a language dependent translate and enable the translate, the language code should be in the MSW of D1 and the LSW should be 1.
- To select, IBM or GEM translates, D1 should be 3, or 5 respectively.

SMS.MPTR

Trap #1 D0=$34

Find Message Pointer

Call parameters		Return parameters	
A1	message code (negative)	A1	pointer to message

Error returns:

always okay

This takes the message code in A1 (which may be an address with the MSB set or it may be the message group + message number negated) and finds the pointer to the message (or to an "unknown error" message).

Trap #1 D0=$35	SMS.FPRM
Find Preferred Module	
Call parameters	Return parameters
D1.L language code or 0	D1 preserved
D2.L car registration (space filled) or 0	D2 preserved
D3.L group number / module type	D3 preserved
Error returns:	
always okay	

This finds the preferred language module of the type and group requested.

Trap #3 D0=$25 IOW.FONT

Set or reset the default system fount

Call parameters		Return parameters	
D1		D1	???
D2	"DEFF"	D2.L	preserved
D3.W	timeout	D3.L	preserved
		D4+	all preserved
A0	channel ID	A0	preserved
A1	base of fount	A1	???
A2	base of second fount	A2	preserved
A3		A3	preserved
		A4+	all preserved

Error returns:

NC not complete
ICHN channel not open

This sets or resets the default system font. Each of the two fount addresses can either be the address of a newly supplied fount, or -1 to keep the current setting, or 0 to select the default font which is inbuilt into the system.

Trap #3 D0=$6

IOB.SUML

Send a string of untranslated bytes

Call parameters		Return parameters	
D1		D1.W	number of bytes sent
D2.W	number of bytes to be sent	D2.W	preserved
D3.W	timeout	D3.L	preserved
		D4+	all preserved
A0	channel ID	A0	preserved
A1	base of buffer	A1	updated pointer to buffer
A2		A2	preserved
A3		A3	preserved
		A4+	all preserved

Error returns:

NC not complete
ICHN channel not open
DVFL drive full

Please refer to section 5.3.3 for details of the special treatment afforded to newlines on the console or screen device.

This trap is similar to IOB.SMUL ($7) but it does not translate the characters. Therefore, the setting of translation tables is ignored as well as the parameter in the device open call (e.g. SERd, SERt, PARd, PARt). A save way of sending graphics data or control codes to the printer, as they will never be translated into other byte patterns.

19.5 SMSQ Cache Handling

19.5.1 Principles

SMSQ is implemented on four distinct hardware platforms with a number of variations using four different MC68000 series processors: MC68000, MC68020, MC68030 and MC68040.
Of these processors, only the MC68000 does not suffer from cache problems.

MC68020
The MC68020 has a single instruction cache which treats supervisor mode addresses as being distinct from user mode addresses. Since there is little, if any, code which is executed in both supervisor mode and user mode, the cache is very small (<100 instructions), and this code is unlikely to be modified, the distinction between supervisor mode and user mode will at worst result in some efficiency.

The instruction cache will need to be cleared whenever executable code is loaded on top of executable code which is already in the cache. As executable code can be LOADed and CALLed or it can be EXECUTED, the instruction cache must be invalidated on every iof.load operation, and, possibly, on every iob.fmul operation. As any IO operation will have enough instructions to completely overwrite the cache, and will usually be called from user mode, there is no serious overhead associated with invalidating the cache on every IO operation.

Executable code can also be set up by programs. It is, therefore, necessary to invalidate the cache on every job activation call, and any call to set up interrupt, polled or scheduled tasks. This will occur automatically if the caches are invalidated on every entry.

Self modifying code in programs should not pose a problem, but the precaution of disabling the caches and suspending the scheduler for a few ticks after starting a job has proved valuable for the MC68040 and should be retained for all processors.

MC68030
The MC68030 has separate instruction and data caches which treat supervisor mode addresses as being distinct from user mode addresses. This seems to be a fundamental design error in the processor which it is necessary to circumvent. The data cache supports only cache write through memory updates. This means that the memory is always up to date with the data cache. The instruction cache will not necessarily be up to date with the memory. Even worse, supervisor mode entries in the cache may not be up to date with user mode entries and vice versa. For operating system code to be able to access data set or modified in user mode (i.e. any output operation and many management operations) it is necessary to invalidate the data cache on every operating system entry.

The instruction cache will need to be cleared whenever executable code is loaded on top of executable code which is already in the cache. As executable code can be LOADed and CALLed or it can be EXECUTED, the instruction cache must be invalidated on every iof.load operation, and, possibly, on every iob.fmul operation. As any IO operation will have enough instructions to completely overwrite the cache, and will usually be called from user mode, there is no serious overhead associated with invalidating the cache on every IO operation.

Executable code can also be set up by programs. It is, therefore, necessary to invalidate the cache on every job activation call, and any call to set up interrupt, polled or scheduled tasks. This will occur automatically if the caches are invalidated on every entry.

Self modifying code in programs should not pose a problem, but the precaution of disabling the caches and suspending the scheduler for a few ticks after starting a job has proved valuable for the MC68040 and should be retained for all processors.

The data cache will also need to be invalidated if there is a DMA access. For external caches, this should be performed automatically by the external cache hardware. The internal caches need to be invalidated on any DMA read operation.

MC68040
The MC68040 has separate instruction and data caches which are accessed by the real address. Unlike the MC68020 and MC68030, code in supervisor mode can read data written in user mode and vice versa. There is, therefore, no need for the caches to be invalidated on every operating system entry.

The MC68040 also provide "snooping" to detect other "bus masters" which may update the memory (e.g. DMA devices). The designers, however, failed to notice that the "Harvard" architecture of the MC68040 requires the implementation of the processor as two separate bus masters, which of course, should require to snoop each other as well as the external bus. (As the instruction unit is a read only bus master, the data unit bus master will, however, never need to snoop the instruction unit.) As a result, the instruction cache will not necessarily be up to date with either the memory or the data cache.

The instruction cache will need to be cleared whenever executable code is loaded on top of executable code which is already in the cache. As executable code can be LOADed and CALLed or it can be EXECUTED, the instruction cache must be invalidated on every iof.load operation, and, possibly, on every iob.fmul operation (this is not done in current versions).

Executable code can also be set up by programs. It is, therefore, necessary to invalidate the cache on every job activate call, and any call to set up interrupt, polled or scheduled tasks.

Self modifying code in programs should not pose a problem, but the precaution of disabling the caches and suspending the scheduler for a few ticks after starting a job has proved valuable for this processor.

The data cache should not need to be invalidated if there is a DMA access: the bus snooping should take care of this.

It is assumed that the data cache will be in write through mode.

MC68060
The cache architecture of the MC68060 is, in most respects, compatible with the MC68040. The branch cache should be handled the same as the instruction cache.

19.5.2 Cache Manipulations

Not all of the fundamental operations are required for cache handling.

Name	Operation	Usage
CINVB	Invalidate both caches	Change from user to supervisor mode
CINVD	Invalidate data cache	Before or after DMA read
CINVI	Invalidate instruction cache	Before executing new code i.e. on resetting vectors on load operations
CDISB	Disable both caches	User CACHE-OFF request
CDISI	Disable instruction cache	Before activating a job
CENAB	Enable both caches	User CACHE_ON request
CENAI	Enable instruction cache	17 ticks after activating a job

Note that either the CDIS or the CENA operations must include a cache disable operation. For simplicity this is included in the CENA operations only.

Most of these operations are performed with one or two MOVEC instructions.

```
$4E7An002 MOVEC    CACR,Dn      Get cache control register
$4E7Bn002 MOVEC    Dn,CACR      Set cache control register
```

The main problem is that the different processors have different organisations of the cache control register

	31	30	29	28	27	23	22	21	15	14	13	12	11	10	9	8	4	3	2	1	0
020																		II	IC	IF	IE
030												DB	DI	DC	DF	DE	IB	II	IC	IF	IE
040	DE								IE												
060	DE	DF	DS	DP	D2	BC	BI	BIU	IE	IF	I2										

Where
- I. is the instruction cache
- D. is the data cache
- B. is the branch cache
- .E is enable when set
- .F is freeze when set
- .C is clear entry when set
- .I is invalidate (clear all) when set
- .IU is invalidate user mode entries when set
- .B is burst access enabled when set
- .S is write store buffer enabled when set
- .P is push without invalidate when set
- .2 is half cache mode when set

The absence of invalidate bits in the MC68040 and MC68060 means that a separate instruction is required for this.

19.5.3 Encoding the Cache Operations

CINVB

MC68020 MC68030			MC68040 MC68060		
$4E7An002	MOVEC	CACR,Dn	Not required		
	OR.W	#$808,Dn			
$4E7Bn002	MOVEC	Dn,CACR			

CINVD

MC68020 MC68030			MC68040 MC68060		
$4E7An002	MOVEC	CACR,Dn	$F458	CINVA	D
	OR.W	#$800,Dn			
$4E7Bn002	MOVEC	Dn,CACR			

CINVI

MC68020 MC68030			MC68040 MC68060		
$4E7An002	MOVEC	CACR,Dn	$F498	CINVA	I
	OR.W	#$8,Dn			
$4E7Bn002	MOVEC	Dn,CACR			

CDISB

MC68020 MC68030			MC68040 MC68060		
	MOVEQ	#0,Dn		MOVEQ	#0,Dn
$4E7Bn002	MOVEC	Dn,CACR	$4E7Bn002	MOVEC	Dn,CACR

CDISI

MC68020		
	MOVEQ	#0,Dn
$4E7Bn002	MOVEC	Dn,CACR

MC68030			MC68040 MC68060		
$4E7An002	MOVEC	CACR,Dn	$4E7An002	MOVEC	CACR,Dn
	CLR.B	Dn		CLR.W	Dn
$4E7Bn002	MOVEC	Dn,CACR	$4E7Bn002	MOVEC	Dn,CACR

CENAB

MC68020 MC68030			MC68040 MC68060		
	MOVE.W	#$1919,Dn	$F4D8	CINVA	DI
$4E7Bn002	MOVEC	Dn,CACR		MOVE.L	#$C0808000,Dn
			$4E7Bn002	MOVEC	Dn,CACR

CENAI

MC68020 MC68030			MC68040 MC68060		
	MOVE.W	#$1819,Dn	$F4D8	CINVA	I
$4E7Bn002	MOVEC	Dn,CACR		MOVE.L	#$C0808000,Dn
			$4E7Bn002	MOVEC	Dn,CACR

19.5.4 Using The Cache Operations

The operating system and device driver code makes no assumptions about the nature of the processor: no cache dependencies are embedded in the code.

CINVB
CINVB is used on all trap #0, #1, #2 and #3 entries. It is implemented as a stub of code before the standard vector entry. For the MC68020 and MC68030 processors, the vector is moved by 10 bytes to include the cache invalidate.

CINVD
A call to CINVD is built into the any device drivers which use DMA. CINVD is implemented as a routine, in the base area, set up for the particular processor.

CINVI
A call to CINVI is built into the IO sub-system for the IOB.FMUL and IOF.LOAD operations. Since all IO operations will have invalidate both caches for the 020 and 030, this is only necessary for the 040 and 060. It is also called by any code which resets executable action routine vectors (e.g. DV3_SETFD). CINVI is implemented as a routine, in the base area, set up for the particular processor.

CDISB
A call to CDISB is built into the "set cache" operating system call. CDISB is implemented as a routine, in the base area, set up for the particular processor.

CDISI
A call to CDISI is built into the "activate job" operating system call. CDISI is implemented as a routine, in the base area, set up for the particular processor.

CENAB
A call to CENAB is built into the "set cache" operating system call. CENAB is implemented as a routine, in the base area, set up for the particular processor.

CENAI
A call to CENAI is built into the polled scheduler entry. CENAI is implemented as a routine, in the base area, set up for the particular processor.

System Variables

sys_castat	$C8	word	MSB set if cache fully enabled
sys_casup	$C9	byte	I cache suppressed timer, counts down to -1

Testing the word sys_castat will yield

NZ	if the caches are enabled or may be enabled,
GT	if the instruction cache is temporarily suppressed,
LT	if the instruction cache is enabled,
Z	if the caches are disabled or there is no cache.

Appendix A Updates & Hints

This appendix contains changes on other system software and standards which are not described in the QDOS Reference Manual. As the update support for the QDOS Reference Manual is the only real update service on any technical QDOS-related printed matter, it now informs you about all kind of changes.

Additional information on WM.ERSTR

The QPTR manual did not mention that there is a limit on own error messages.
An own error messages is easy to create:

```
    LEA       own_msg,A0     ; get address
    MOVE.L    A0,D0          ; into our "error" register
    BSET      #31,D0         ; an error is negative
```

Now the limit: the length of the string is limited to 40 ($28) characters. If it is longer, "unknown error" is returned instead!

Additional information on WM.LDRAW

WM.LDRAW clears the change bit in the status are of every item which is selectively redrawn.

Additions to the CONFIG standard

The attributes for strings have been extended, to allow menu-driven CONFIG programs better options for a selection, depending on the type. There are two additional bits used in the string attributes: 8 and 9. These define the type of string, so that the CONFIG program can treat these strings in a special way. The possible combinations are:

cfs.sspc	%0000000000000001	string strip spaces
cfs.file	%0000000100000000	string is filename
cfs.dir	%0000001000000000	string is directory
cfs.ext	%0000001100000000	string is extension

At present, these features are supported by the new MenuConfig, and ignored by the standard config.

Additions to IOP.RPTR and Pointer Record

Bits 23 to 8 of the event vector in the pointer record are already used by the Window Manager. The 8 job events are, therefore, mapped into the most significant 8 bits (pp_jevnt) of the event vector within the pointer record and for the IOP.RPTR operating system call.

Note that while all pointer events that have occurred since the call are filled into pt_pevnt in the pointer record, only those job events (including pending events) which caused the return are filled into pt_jevnt.

New Pointer Event

Pointer event bit 6 (number 64) is now used to indicate that the pointer sprite has hit the edge of the screen.