QDOS / SMS / SMSQE

Reference Guide

Version 4.8

Foreword

This is the 4th edition of the QDOS/SMS Reference Manual, a guide and manual for programming the QL as well as QDOS and its many descendants, including especially SMSQ/E. The purpose of this is to have an up-to-date guide to the facilities offered by QDOS and SMSQ/E. This text is based on the original 3rd edition of the manual, as it was published by Jochen Merz and Marcel Kilgus, and then scanned in by Derek Stewart. Except to make changes for error corrections and new insertions, I left much of the original text untouched, even where it was mostly outdated.

Of course, all of this was made possible by the original writers of the original texts (Tony Tebby, Jochen Merz, Marcel Kilgus), and thanks go to them.

As to the amendments made in this text, I did correct all errors I was able to spot. I also continued to point out the differences between the plain QDOS variety of things and those for SMSQ/E, where appropriate. Included in this edition are the updates for SMSQ/E as they stand now. It is true that this text now contains much information that is specific to SMSQ/E, but this is due to the fact that SMSQ/E is still being developed, whilst the other OSes aren't. Thus, there are now sections on the HOME thing, the SMSQ/E style guide etc.

In this manual, S*Basic means the QL's SuperBASIC and SMSQ/E's SBasic. SMSQDOS means something is applicable to SMSQ/E and QDOS. Sometimes you will find reference to assembler key files (e.g. keys_qdos_io). These refer to the keys files as found in the "keys" subdirectory in the SMSQ/E sources.

The page numbers in each section and indexes thereto refer to the page numbers of that section. Unfortunately, whilst the initial table of content is " clickable" (CTRL + left mouse click on a section to go there), this is not true for the indexes.

Much care has gone into trying to make sure that the information herein is correct. All remaining errors/omissions are mine.

Ideas, corrections and / or suggestions are always welcome. After version 4.0, this manual is maintained by W. Lenerz only. Per Witte has pointed out numerous improvements/errors.

Wolfgang Lenerz
Derek Stewart

Versions of this manual as of v. 4.1 (all by WL)

v. 4.8 Some typos (6-2, 6-4, 16-7, 13-1), removed reference to system variable **sys_10i** (was at $00c2). A value of -1 in D3 is "delete" in IOA.OPEN, some explanation on IOA.DELF in SMSQ/E. Correct examples for vector IOU.SSIO. Typos/error corrections and additions in Appendix A. Vector MEM.ALHP: the condition code is not cleared on success on all QDOS ROM versions (it is on SMSQ/E). Added order of keyboard tables.

v. 4.7 Better explanation of the value retuned in D1 by SMS_ACHP and of how literal numbers are stored in S*Basic. Included the table for SMSQ/E Sbasic variables.

v. 4.6 The length word in D2 for IOB.FLIN and IOB.FMUL is a positive word. SBasic arithmetic operation **qa.flong** renamed to **qa.fltli** to keep in sync with the `keys_qlv` keys file. Correct reference to language handling for trap#1 with **sms.ldmm** and following. CV.DATIL is available on SMSQ/E machines only; IOB.EDLIN is really called IOB.ELIN.

v. 4.5 Typos, added keys/information about standard hard disk format, SBasic name table additions for integer REPeat and FOR loop indexes, warning for device driver linkage blocks**,** IOB.FLIN under SMSQ/E level 3 drivers may convert <CR><LF> to <LF>.

v. 4.4 Wrong label for SMS.LSHD corrected in trap description. MEM.ACHP does not modify A3 in SMSQ/E. Entry regs to vector $11C corrected. Correct registers for vector SB.GTINT and following shown. Current thing parameters completed and some keys don't exist. On SMSQ/E it is not necessary to have the maths stack pointer in A1 before calling vector QA.RESRI. Spurious content of D2 & D3 removed from CV.ILDAT and CV.ILDAY.

v. 4.3 Added some hyperlinks. IOU.DNAM: corrected spelling of some examples. Explained that opening a directory will open the next higher directory if not found. Typo corrections in the hardware keys section. Added keys for pointer device.

v. 4.2 Corrected wrong register on entry to SB.PUTP (was A1, is now A3). IOB.SMUL: the buffer size is a positive word; D1 upper word is destroyed. IOB.FLIN/IOB.SMUM: error return if no LF found corrected, must be ERR.BFFL and not ERR.OVFL; D1 upper word is destroyed. IOU.DNAM correct title for trap. RCNT_GARJ, RCNT_GALL, RCNT_GALJ : return description, parameter description and examples corrected.

v. 4.1 Corrected missing source & destination registers when restoring SR in section 10.9 example.

The page numbers in this clickable table of contents refer to the page numbers within each section. CTRL+Left Click goes to the entries.

Table of contents

0. Why this book? (Original foreword by Jochen Merz)....1
1. About this Guide....1
2. Introduction to QDOS / SMS / SMSQ/E....1
2.1. Memory Map....1
2.1.1. Principles....2
2.1.2. System Variables....2
2.1.3. System Management Tables....2
2.1.4. Common Heap Area....2
2.1.5. Free Memory Area....2
2.1.6. S*Basic area....3
2.1.7. Transient Program Area....3
2.1.8. Resident Procedure Area....3
2.2. Calling QDOS/SMS Routines....3
2.2.1. Traps....3
2.2.2. Vectored Routines....4
2.2.3. Atomic Actions....5
2.3. Exception Processing....5
2.4. Start-up....6
3. Machine Code Programming....1
3.1. Jobs....1
3.1.1. Normal Jobs....1
3.1.2. Special Programs....3
3.1.3. Job Control Enhancements [SMSQ/E]....4
3.2. S*Basic Procedures and Functions....5
3.3. Tasks....5
3.4. Operating System Extensions....5
4. Memory Allocation....1
4.1. Heap Mechanism....1
5. Input/ Output on the QL....1
5.1. Serial I/O....1
5.2. File I/O....2
5.3. Screen and Console I/O....3
5.3.1. Display Modes....3
5.3.2. Window Properties and Operations....3
5.3.3. Screen Character Output Operations....5
5.3.4. Graphics Operations....5
5.3.5. Special Properties of Console Channels....6
5.3.6. Special Keyboard Functions....6
5.3.7. Extended Operations [SMSQ/E]....6
5.3.8. Display [SMSQ/E]....6
5.3.8.1. New CON driver vectors....6
5.3.8.2. New (WMAN) colour format....14
5.3.8.2.1. Stipple Format....14
5.3.8.2.2. 3D Border Format....14
5.3.8.3. System palette entries....15
5.3.8.4. New Basic Keywords....16

5.3.8.4.1. Colours....16
5.3.8.4.2. Palette handling....17
5.3.8.4.2.1. System palette keywords....17
5.3.8.4.2.2. Job palette keywords....17
5.3.8.5. New Move modes....18
5.3.8.5.1. The move modes....18
5.3.8.5.2. Configuring/setting the move mode....18
5.3.8.5.3. Configuring/setting the degree of transparency....19
5.3.8.6. Graphics with alpha blending....20
5.3.8.6.1. Machine code interface....20
5.3.8.6.2. S*Basic keywords....21
6. QDOS Device Drivers....1
6.1. Device Driver Memory Allocation....2
6.2. Device Driver Initialisation....2
6.3. Physical Layer....3
6.3.1. External Interrupt Tasks....3
6.3.2. Polling Interrupt Tasks....3
6.3.3. Scheduler Loop Tasks....3
6.4. The Access Layer....4
6.4.1. The Channel Open Routine....4
6.4.2. The Channel Close Routine....5
6.4.3. Input/Output Routine....6
7. Directory Device Drivers....1
7.1. Initialisation of a Directory Driver....2
7.2. Access Layer....3
7.2.1. The Channel Open/File Delete Routine....4
7.2.2. The Channel Close Routine....5
7.2.3. The Input/ Output Routine....6
7.3. Slaving....6
7.4. The Format Routine....8
8. Built-in Device Drivers....1
8.1. QL Floppy Disc Format [EXT]....2
8.2. Direct Sector Read/Write [EXT]....4
8.3. Additional Standard Device Drivers [ST] [EXT] [SMSQ/E]....4
9. Interfacing to S*Basic....1
9.1. Memory Organisation within the S*Basic Area....1
9.2. The Name Table....2
9.3. Name List....3
9.4. Variable Values Area....3
9.5. Storage Formats....3
9.5.1. Integer Storage....3
9.5.2. Floating Point Storage....3
9.5.3. String Storage....3
9.5.4. Array Storage....3
9.6. Code Restrictions....4
9.7. Linking in New Procedures and Functions....4
9.8. Parameter Passing....5
9.9. Getting the Values of Actual Parameters....5
9.10. The Arithmetic Stack Returned Values....6
9.11. The Channel Table....6
10. Hardware-related Programming....1
10.1. Memory Map [QL]....1
10.2. Display Control....2
10.3. Display Control Register....2
10.4. Keyboard and Sound Control....2

10.5. Serial I/O....3
10.6. Real-time Clock....3
10.7. Network....3
10.8. Microdrives....3
10.9. User and Supervisor Mode [ST]....5
10.10. The Interrupt System [ST]....6
10.11. The MIDI Interrupt server [ST]....7
10.12. Different Processors [ST][SMSQ/E]....7
10.13. Different Machines [ST, SMSQ]....7
10.14. The ATARI DMA [ST]....8
11. Adding Peripheral Cards to the QL....1
11.1. Expansion Connector....1
11.2. CPU Interface....1
11.3. Peripheral Card Addressing....2
11.4. Add-on Card ROMs....2
12. Non-English Systems....1
12.1. Video....1
12.2. Non-English-language Keyboards....1
12.3. Character Set [not SMS2] [SMSQ]....2
12.4. Special Alphabets....3
13. System Traps....1
13.1. Trap 1 Keys - numerical order with page reference....22
14. I/O Management Traps....1
14.1. Trap 2 Keys - numerical order with page reference....6
15. I/O Access Traps....1
15.1. Trap 3 Keys - numerical order with page reference....38
16. Vectored Routines....1
16.1. Vectored Routines - numerical order with page reference....22
17. Things [EXT][SMSQ/E]....1
17.1. Thing structures....2
17.1.1. Thing linkage format....2
17.1.2. Thing header format....2
17.1.3. List of Things Header....2
17.1.4. Executable Thing Header....3
17.1.5. Extension Thing Header....3
17.2. Different sorts of Thing....3
17.3. Thing vectors....4
17.4. Thing Entry Points....11
17.4.1. TH_ENTRY....11
17.4.2. TH_EXEC....11
17.4.3. Example of entries to the Thing Vector system....11
17.5. Extension Things....14
17.5.1. Extension Thing Header....14
17.5.2. Level 1 Extension Thing Parameter Definition....14
17.5.3. Call Values and Keys....15
17.5.4. Pointer Parameter Usage....15
17.5.5. Optional Parameter....16
17.5.6. Array Parameter....16
17.5.7. Parameter Types....16
17.5.8. Example Parameter Definitions....16
17.5.9. Parameter List....17
17.5.10. Defining Extension Things....17
17.5.11. Accessing Extension Things....17
17.5.12. When to Use Extension Things....17

17.6. Thing-supplied code....18
18. Keys....1
18.1. Error keys....1
18.2. System variables....2
18.3. SuperBasic Variables....7
18.4. SBasic Variables [SMSQ/E]....10
18.5. Basic channel definitions and tokens....12
18.5.1. Offsets on BASIC Channel Definitions....12
18.5.2. BASIC Token Values....12
18.6. Job Header and Save Area Definitions....15
18.7. Slave Memory Block Table Definitions....16
18.8. Channel Definitions....17
18.9. File System Definition Blocks....18
18.9.1. 18.Standard channel block for filing system....18
18.9.2. The common part of a physical definition block....18
18.9.3. Microdrive Physical Definition Block [QL]....19
18.9.4. Other Filing System Physical Definition Block [SMSQ][EXT]....19
18.10. Device Driver Linkage Block....20
18.10.1. Screen Driver Data Block Definition....21
18.10.2. Serial Channel Definition Block [QL]....22
18.10.3. Network Channel Definition Block [QL]....22
18.11. Queue Header Definitions....22
18.12. Arithmetical Interpreter Operation Codes....23
18.13. IPC Link Commands....24
18.14. Hardware Keys....25
18.15. Trap Keys....27
18.15.1. Trap 1 Keys (System Traps)....27
18.15.2. Trap 2 Keys (I/O Allocation Traps)....28
18.15.3. Trap 3 Keys (I/O Traps)....29
18.16. List of Vectored Routines....31
18.17. Keys for Things....33
18.18. Keys for HOTKEY Thing....36
18.19. Keys for format of pointer device driver definition block....37
18.20. Hard disk format: QLWA....41
19. SMSQ....1
19.1. Language handling in SMSQ....1
19.1.1. Principles....1
19.1.2. Classification of Language Dependent Modules....1
19.1.2.1. Printer Translate Tables....1
19.1.2.2. Keyboard Tables....1
19.1.2.3. Message Tables....2
Language Preference Tables....2
19.1.3. Language Dependent Module Structure....3
19.1.4. Language Specification....3
19.1.5. Implementation....3
19.1.6. System Variables....3
19.1.7. Additional Trap #1 Calls....4
19.2. Additional Trap #3 calls....7
19.3. SMSQ Cache Handling....7
19.3.1. Principles....7
19.3.1.1. MC68020....7
19.3.1.2. MC68030....8
19.3.1.3. MC68040....8
19.3.1.4. MC68060....9
19.3.2. Cache Manipulations....10

19.3.3. Encoding the Cache Operations....11
19.3.4. Using The Cache Operations....12
19.3.4.1. CINVB....12
19.3.4.2. CINVD....12
19.3.4.3. CINVI....12
19.3.4.4. CDISB....12
19.3.4.5. CDISI....12
19.3.4.6. CENAB....12
19.3.4.7. CENAI....13
19.3.4.8. System Variables....13
20. The HOTKEY System II [EXT]....1
20.1.1. The HOTKEY Item....6
20.1.2. Hotkey Vectors....6
21. The Button Frame [EXT]....1
22. The HOME Thing [EXT] [SMSQ/E]....1
22.1. Purpose and facilities....1
22.1.1. Home directory....1
22.1.2. Home Filename....1
22.1.3. Current Directory....1
22.1.4. Default Directory for named jobs....1
22.2. The HOME Thing under SMSQ/E and QDOS....2
22.2.1. SMSQ/E....2
22.2.1.1. The EX(ec) etc commands....2
22.2.1.2. QPAC II and other file managers....2
22.2.1.3. FileInfo....2
22.2.1.4. Basic....2
22.2.2. QDOS....2
22.3. Using the HOME Thing....3
22.3.1. From SBasic....3
22.3.1.1. Get the home directory....3
22.3.1.2. Get the home filename....3
22.3.1.3. Get the current directory....3
22.3.1.4. Default names....3
22.3.1.5. Get the version of the HOME Thing....3
22.3.2. From machine code....4
22.4. Setting up a home directory....6
22.4.1. From S*Basic....6
22.4.2. From Machine Code....6
23. The RECENT Thing [SMSQ/E]....1
23.1. Concepts....1
23.1.1. The lists....1
23.1.2. Job IDs....2
23.1.3. Buffers....2
23.2. The Thing interface in Assembler....3
23.2.1. JobIDs and Name Pointer....4
23.2.2. The extensions....4
23.3. SBasic keywords....14
23.4. Configuration....21
23.5. Performance penalty....21
24. Appendix A Compiling SMSQE with SMSQEMake....1
24.1. Compiling the source code....1
24.2. Requirements....1
24.2.1. The DEV device....1
24.2.2. The assembler....2
24.2.3. The linker, cctf and make programs and how to use them....2

24.2.3.1. The Make Program....2
24.2.3.2. The linker....2
24.2.3.3. CCTF....2
24.3. How to use SMSQEMake....3
24.3.1. Setting up the environment....3
24.3.2. Description of the program....3
24.3.2.1. The title bar....3
24.3.2.2. The target row....3
24.3.2.3. The link files window....3
24.3.2.4. The "All" item....3
24.3.2.5. The "OK" item....3
24.3.2.6. The DEL item....3
24.3.2.7. The "Make" item....4
24.3.3. Command line parameters....4
24.3.4. A proposed way of working....5
24.3.5. Error reports....5
24.4. Recompiling or changing SMSQEMake....5
24.5. Additional programs....5
25. Appendix B Official SMSQ/E style guide....1
25.1. Generic requirements....1
25.1.1. Development system....1
25.1.2. Assembler....1
25.1.3. Character set....1
25.1.4. TAB stops....1
25.2. Assembler files....2
25.2.1. Generic file structure....2
25.2.2. Headers....3
25.2.3. Cases....3
25.2.4. Comments....3
25.2.5. Labels....4
25.2.6. References to include and other files....4

0. Why this book? (Original foreword by Jochen Merz)

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 or other emulators. 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 (/E) 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.
[SMSQ/E]	Needs the operating system SMSQ/E to be installed, preferably in the most recent version.

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. 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 firm-ware 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. Introduction to QDOS / SMS / SMSQ/E

QDOS is the QL operating system. SMS is an advanced version, completely reprogrammed but as compatible as possible. SMSQ/E is the modern evolution of SMS. All of them are single-user multi-tasking operating systems: that is, they provide the means for several independent programs to run concurrently; in the QL or elsewhere, but do 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.

In most cases in this book, wherever QDOS is mentioned, the explanation also applies to SMSQ/E, if not, this will be stated.

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 S*Basic 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 S*Basic 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. SMSQ/E supports much more memory, in theory it can address the whole 32 bit memory. However, since some programs, notably Qliberator, use the upper 3 bits of addresses for their own purpose, most SMSQ/E machines will limit this to something like 256 MB. The memory map is as follows:

SYS_RAMT	Top of RAM	
SYS_RPAB	Resident procedure area	
SYS_TPAB	Transient program area	
SYS_SBAB	S*Basic 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 S*Basic 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).

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). 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 section 4.1.

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 S*Basic area), special provisions are made so that all three can grow at the appropriate times.

2.1.6. S*Basic area

The S*Basic 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 S*Basic 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 S*Basic interpreter if an area above it needs to grow, or if the S*Basic 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 S*Basic 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 S*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 S*Basic interpreter divides its working area into several portions, details of which may be found by looking at the **BV** definitions in Section 18.3. (for QDOS) and the **SB** definitions in section 18.4 (for SMSQE) All of the pointers to these various portions are also relative to A6.

Note that, under SMSQ/E, the SBasic area doesn't move. If you write an extension, references thus needn't be relative to A6 during the entire processing. However, doing so will make your extension incompatible with QDOS.

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 S*Basic 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 S*Basic 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 18.1 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 SMSQDOS 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 system traps are given in Sections 13 – 15.

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 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. Under SMSQ/E, 512 bytes may be used on the supervisor stack.

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=0) 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 through 15.

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.

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. Under QDOS (not SMSQ/E), 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 S*Basic area.

The address **$C000** is then checked by QDOS for the characteristic longword **$4AFB0001**: if this is found, QDOS links in the S*Basic 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 S*Basic 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 S*Basic program. SMSQ/E will search for either “FLP1_BOOT” or “WINx_BOOT”, according to the way it is configured. If both of these attempts fail, the system starts up the S*Basic interpreter with an empty program memory.

3. Machine Code Programming

Five types of machine code are available to program the QL, each being used to perform quite different operations: jobs, S*Basic procedures and functions, tasks, the operating system or extensions to it and “Things”. Thus there are several differences in both the form in which they are written, and the way in which they are treated by QDOS. Things have their own section in this manual.

3.1. Jobs

3.1.1. Normal 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 SMSQ/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 long words 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 S*Basic **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 entry point 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 long words each of which points to a job control block: the contents of this are described in Section 18.6. 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).

3.1.2. 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 S*Basic 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.

A0 points to two support routines, the first lies a (A0) and gets a filename, the second lies at 2(A0) and opens a channel:

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 S*Basic 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 default directory in front). If the open fails, A1 will point to the default+given filename. The channel ID is returned in A0 and all other registers are preserved.

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

3.1.3. Job Control Enhancements [SMSQ/E]

The S*Basic extensions FEX, FEW, FET and FEP have been added to SMSQ/E v3.00 and later.

These are function calls corresponding to the procedures EX (EXEC), EW (EXEC_W), ET and EXEP.

FEX

```
job_id = FEX(<file name>)
```

Executes and returns the ID of the job <file name>.

This ID can be used to manipulate the job in various ways by using the other job control extensions, such as SPJOB, AJOB, RJOB, etc.

The full syntax using input and output channels, as well as filters, is supported. See the TK2 documentation, Section 8.xx for details.

Note: In the event of filters being set up, only the ID of the first job is returned.

Note: The name FEX clashes with the eponymous keyword from FileInfo2. By the time you read this a later version of FI2 may be available, otherwise you will need to patch one or the other of the keywords to access both.

FET

As for FEX above, except the job is not activated.

FEW

```
er = FEW(<file name>)
```

Returns the error code returned by the (first) job. Syntax as for FEX above.

Note: FEW tries to open the channels of files supplied in the parameter list before executing the job(s). Any errors arising from this, including erroneous parameters, are returned to the caller as "hard" errors.

FEP

```
job_id = FEP(<thing name>)
```

Executes and returns the ID of the job <thing name>. FEP is the implementation of EXEP as a function. Refer to your Qpac2 manual for details.

EXF

```
job_id = EXF (<file_name>)
```

This keyword is, in function, totally identical to the FEX keyword introduced by version 3.00 of SMSQ/E (it uses the same code, just another name).

The FEX keyword in SMSQ/E FEX clashes with the FEX keyword contained in FileInfo II. To avoid having to patch either SMSQ/E or FileInfo II (even though an S*Basic program to patch FileInfo II is supplied) you can simply use the **EXF** keyword, instead of the SMSQ/E **FEX** keyword.

3.2. S*Basic Procedures and Functions

The S*Basic 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 S*Basic 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 either job number zero (in QDOS) or any other S*Basic job (under SMSQ/E), 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 S*Basic data area. On the QL under QDOS, 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 stack pointer A7 must of course be restored to its original value before exiting from the procedure. Note : this restriction concerning register A6 does not apply to SMSQ/E.

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 S*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 S*Basic 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 S*Basic 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 S*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 to 8).

4. 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 S*Basic 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 S*Basic 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 free heap items. Each heap item is an area of memory (which is a multiple of 8 bytes long), together with a pair of long words: 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. Note that memory is always allocated as a multiple of 8 bytes.

- 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.

- The vectored routines **MEM.ALHP** and **MEM.REHP** may be used for allocating/releasing memory within a heap.

5. 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 recognise 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 S*Basic channel number (denoted by #expression): the latter is the index of an entry in the S*Basic 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 port 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 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.CHEK** commands are subtly different: **IOF.FLSH** ensures that all write operations are complete, whereas **IOF.CHEK** ensures that all write and read operations (including pre-fetches) are complete.

Not all drivers will implement this trap, e.g. for the SMSQ/E inbuilt ram disks, where this will just go to a MOVEQ #0,D0 and an RTS.

SMSQ/E contains several additional operations for filing system devices. Most filing system devices under SMSQ/E will allow these operations :

IOF.RNAM	rename file
IOF.TRNC	truncate file to current position
IOF.DATE	set or get file dates
IOF.MKDR	make directory
IOF.VERS	set or get version
IOF.XINF	get extended information

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 S*Basic 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.**

SMSQ/E has many more display modes, which ones can be displayed depends on the machine it is running on.

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 (mode 8) and for several commands in 512-pixel mode (mode 4), 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. Under SMSQ/E, or under QDOS with the pointer environment, overlapping windows are restored by the system.

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

SMSQ/E has many more window traps, some of these will also be available under QDOS with the pointer environment:

EXTENDED COLOUR TRAPS

IOW.PAPP	define paper colour (palette)
IOW.STRP	define strip colour (palette)
IOW.INKP	define ink colour (palette)
IOW.BORP	define border (palette)
IOW.PAPT	define paper colour (24 bit)
IOW.STRT	define strip colour (24 bit)
IOW.INKT	define ink colour (24 bit)
IOW.BORT	define border (24 bit)
IOW.PAPN	define paper colour (native)
IOW.STRN	define strip colour (native)
IOW.INKN	define ink colour (native)
IOW.BORN	define border (native)
IOW.BLKP	draw block (palette)
IOW.BLKT	draw block (24 bit)
IOW.BLKN	draw block (native)
IOW.PALQ	define QL colour palette
IOW.PALT	define 8 bit palette
IOW.SALP	set alpha blending weight

POINTER I/O TRAP KEYS

IOP.WPAP	define wallpaper
IOP.FLIM	Find window LIMits
IOP.SVPW	SaVe Part of Window
IOP.RSPW	ReStore Part of Window
IOP.SLNK	Set bytes in LiNKage block
IOP.PINF	pointer information
IOP.RPTR	read pointer
IOP.RPXL	read pixel

IOP.WBLB	write blob
IOP.LBLB	write line of blobs
IOP.WSPT	write sprite
IOP.SPRY	spray pixels
IOP.FILM	fill within mask
IOP.SPLM	set pointer limits
IOP.OUTL	set window outline
IOP.SPTR	set pointer position
IOP.PICK	pick / bury window
IOP.SWDF	set window definition
IOP.WSAV	locate and save window
IOP.WRST	restore window

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 S*Basic 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 [SMSQ/E]

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 (I/O Traps) for a more detailed discussion of this trap.

5.3.8. Display [SMSQ/E]

This section documents many of the enhancements to SMSQ/E v3.00 and following, mostly directed at programmers.

5.3.8.1. New CON driver vectors

A new vector block has been introduced to provide direct access to new screen driver functions. To call one of those functions, one first needs a pointer to the CON linkage block. This can either be obtained in the traditional way or by reading the **sys_clnk** ($C4) system variable. It is planned that future PTR_GEN / WMANs for non-SMSQ/E version will also support this system variable. On current non-SMSQ/E systems its value should be 0.

The pointer to the vector table itself is located in the new **pt_vecs** variable within the linkage block. A typical call sequence can thus look like this:

```
moveq       #sms.info,d0
trap        #1                ; get pointer to system variables in A0
move.l      sys_clnk(a0),a3   ; pointer to CON linkage
move.l      pt_vecs(a3),a0    ; vector table
jsr         pv_fspr(a0)       ; actual call
```

All vectors expect A3 to be the pointer to the CON linkage block on entering the call. With the above code, this is done automatically. The keys (e.g. the values of **PV_PINF**, **PV_FSPR** etc) are contained in the file "dev8_keys_con".

Vector $00

PV_PINF

Like IOP.PINF, but one doesn't need a channel to call this routine.

Call parameters		Return parameters	
D1		D1	Pointer Version Number
D2		D2	Preserved
D3		D3	Preserved
A0		A0	Preserved
A1		A1	Pointer to WMAN
A2		A2	Preserved
A3	Pointer to CON Linkage Block	A3	Preserved

Error returns:

This routine always succeeds.

Vector $06

PV_FSPR

Look in linked sprite list for the definition that would actually be used in the current display mode.

Call parameters		Return parameters	
D1		D1	Preserved
D2		D2	Preserved
D3		D3	Preserved
A0	Pointer to 1st Sprite	A0	Pointer to Fitting Sprite
A1		A1	Pointer to WMAN
A2		A2	Preserved
A3	Pointer to CON Linkage Block	A3	Preserved

Error returns:

This routine always succeeds.

If no fitting sprite is found, a pointer to the arrow sprite is returned!

Vector $0C

PV_SSPR

Set system sprites/Get system sprite address

Call parameters		Return parameters	
D1.W	Sprite Number / -ve	D1	Preserved / Max Allowed | Max Current
D2		D2	Preserved
D3		D3	Preserved
A0		A0	Pointer to Fitting Sprite
A1	Pointer to Sprite / 0	A1	Preserved / Pointer to Sprite
A2		A2	Preserved
A3	Pointer to CON Linkage Block	A3	Preserved

Error returns:

IPAR	Illegal sprite number (set / get)
ITNF	there are no system sprites !

This gets or sets a system sprite or returns the maximum number of system sprites

- If D1 is a negative number (-1 is suggested), then on return d1 contains:
  maximum number of space in table for system sprites | highest number of current system sprite

else:

- If A1 = 0, then
  one gets the address of the system sprite the number of which is passed in D1. The address is returned in a1. This address MAY be 0, in which case the system sprite requested does not exist. This will only happen if somebody fiddled with the table contrary to recommendations

- If A1 <> then
  it contains the address of a sprite that will be a system sprite, d1 contains the number of that sprite. This sprite is not " copied to a safe place", it is the responsibility of the calling job to make sure that the sprite doesn't just disappear

For a list of the system sprites defined thus far see **KEYS_SYSSPR**.

The sprite table has the following format:

-2	maximum number of sprites possible in table (word)
0	number of sprites currently in table (word)
2+	long word absolute pointers (i.e real addresses of sprites)

Vector $12

PV_SIZE

Get shift sizes

Call parameters		Return parameters	
D0		D0	PT.SPXLW | PT.RPXLW
D1		D1+	Preserved
D2		D2	Preserved
D3		D3	Preserved
A0		A0	Preserved
A1		A1	Preserved
A2		A2	Preserved
A3	Pointer to CON Linkage Block	A3	Preserved

D0 returns :

PT.SPXLW : shift pixels to long word

PT.RPXLW : round up pixels to long word

Error returns:

None, this vector always succeeds. The value in D0 is not an error return

Vector $18

PV_MBLK

Moves a block of screen memory about

Call parameters		Return parameters	
D0		D0	Smashed (undefined)
D1	Size of Section to move	D1	Smashed
D2	Old origin in source area	D2	Smashed
D3	New origin in destination area	D3	Smashed
D4		D4	Smashed
D5		D5	Smashed
A2	Row increment of source area	A2	Smashed
A3	Row increment of destination area	A3	Smashed
A4	Base address of source area	A4	Smashed
A5	Base address of destination area	A5	Smashed

All other registers are preserved

Error returns:

This routine always succeeds. The value in D0 is not an error return

This moves a block of screen memory about, from source to destination. The X | Y size of the block, in pixels, is contained in D1 on entry. **Note:** Do not mis-use this vector to move general memory about. The size of the memory actually moved depends on the screen driver that is being used. Thus, if you move a block of 10x20 pixels (x|y size) in modes 32 and 33, 400 bytes will be moved (1 pixel = 2 bytes). In modes 16 and 31, only 200 bytes will be moved (1 pixel = 1 byte) and in the QL modes, even less bytes will be moved.

Vector $1E

PV_CURSP

Sets the per job cursor

Call parameters		Return parameters	
D0		D0	Error
D1	Job ID	D1	Smashed
D2	Status Wished (0 | 1)	D2+	All preserved
A0		A0	Preserved
A1		A1	Preserved
A2		A2	Preserved
A3	Pointer to CON Linkage Block	A3	Preserved

Error returns:

IJOB	Wrong Job ID
NIMP	Something went horribly wrong : no job table!

Please see the Section 5.3.9 : Cursor Sprite for further explanations on this vector.

Vector $24

PV_BGCTL

Gets/sets the background I/O status

Call parameters		Return parameters	
D0		D0	Standard Error Code
D1	-1 read	D1	0 Disabled
	0 disable		>0 Enabled
	1 enable		
D2		D2	Preserved
D3		D3	Preserved
A0		A0	Preserved
A1		A1	Preserved
A2		A2	Preserved
A3	Pointer to CON Linkage Block	A3	Preserved

Error returns:

IPAR	D2 is not 0
NIMP	Operating System is not background I/O compatible

This sets or gets the background I/O status.

If D1 is negative on entry, the current background I/O status is returned, else the current background I/O status is set according to the value of D1 (any value other than 0 enables background I/O).

Vector $2A

PV_CMBBLK

Combines two blocks of (screen) memory with alpha blending and puts the result into the destination block

Call parameters		Return parameters	
D0		D0	Smashed (undefined)
D1	Size of block to combine	D1	Smashed
D2	Origin in source area 1	D2	Smashed
D3	New origin in destination area	D3	Smashed
D4	Origin in source area 2	D4	Smashed
D5		D5	Smashed
D6	Alpha value	D6	Preserved
D7	Row increment of source area 2	D7	Smashed
A1	Base address of source area 2	A1	Smashed
A2	Row increment of source area 1	A2	Smashed
A3	Base address of source area 1	A3	Smashed
A4	Pointer to CON Linkage Block	A4	Smashed
A5	Base address of destination area	A5	Smashed

All other registers are preserved

Error returns:

This routine always succeeds.

The value in D0 is not an error return

This will combine the pixels of two blocks of screen memory with an alpha blending operation and put the resulting block into the destination. The x|y size of the block, in pixels, is contained in D1 on entry. D6 contains the alpha value, from 1 (nearly transparent) to 255 (totally opaque), in the LSB.

NOTE 1: This vector is only implemented for screen modes where alpha blending actually makes sense, i.e. modes 16, 32 and 33.

In other screen modes, such as the QL screen modes, or Atari mono modes, this vector is redirected to vector **PV_MBLK**.

NOTE 2: Do not mis-use this vector to combine general memory.

The size of the memory actually combined depends on the screen driver that is being used. Thus, if you combine a block of 10x20 pixels (x|y size), in mode 16, then 200 bytes will be combined (1 pixel = 1 byte). But in modes 32 and 33, 200 words (400 bytes) will be combined (1 pixel = 2 bytes).

Display Vectors

PV_BGCTL....11
PV_CMBBLK....12
PV_CURSP....11
PV_FSPR....7
PV_MBLK....10
PV_PINF....7
PV_SIZE....9
PV_SSPR....8

5.3.8.2. New (WMAN) colour format

The latest versions of WMAN, the Pointer Environment Window Manager ,contain the possibility to use new colour format. Whilst, strictly speaking, this is a WMAN function, these colours can also be used in non-PE programs under SMSQ/E. Hence the inclusion in this manual.

Colours for the new WMAN are always given as one word. The word may have any of the following formats:

```
%00000000cccccccc exactly as before
%00000001pppppppp palette
%00000010pppppppp system palette
%00000011gggggggg gray scale
%00000100cc00tttd 3d border (border calls only!) see below
%01ssxxxxxxyyyyyy palette stipple see below
%1rrrrgggggbbbbb 15 bit RGB
```

5.3.8.2.1. Stipple Format

s = Stipple code (0 = dot, 1 = horizontal, 2 = vertical, 3 = checkers)
x = Stipple colour
y = Main colour

As x and y can only hold 6 bits, only the first 64 entries of the palette can be used for stippling. Due to the design of the palette those entries alone still cover the whole colour range quite well.

5.3.8.2.2. 3D Border Format

d = Direction (0 = raised, 1 = lowered)
t = Type
c = Compatibility mode

To see what types are available have a look at this image:

The compatibility modes are available on some border types and they tell how to squeeze a non-standard border size into a QL border. Some modes paint areas with the current paper colour, therefore it is a wise idea to always set the paper colour before the border. The WMAN routines have already been changed to take this into account.

In case of a non-standard border width another border call on this window MUST be made through the WMAN routines instead of the standard border calls (e.g. by calling **WM.TRAP3**). Otherwise the overall window size will be altered.

The colours to paint the border are defined in the system palette (**SP.3DDARK** and **SP.3DLIGHT**).

Future versions may shade the paper colour, therefore it's again a good idea to set the paper colour before the border call.

5.3.8.3. System palette entries

The keys for this are defined in the file dev8_keys_syspal.

Please note that you can configure SMSQ/E to set the palette(s) to your taste.

Name	Number	Meaning
SP.WINBD	$0200	Window border
SP.WINBG	$0201	Window background
SP.WINFG	$0202	Window foreground
SP.WINMG	$0203	Window middleground
SP.TITLEBG	$0204	Title background
SP.TITLETEXTBG	$0205	Title text background
SP.TITLEFG	$0206	Title foreground
SP.LITEMHIGH	$0207	Loose item highlight
SP.LITEMAVABG	$0208	Loose item available background
SP.LITEMAVAFG	$0209	Loose item available foreground
SP.LITEMSELBG	$020a	Loose item selected background
SP.LITEMSELFG	$020b	Loose item selected foreground
SP.LITEMUNABG	$020c	Loose item unavailable background
SP.LITEMUNAFG	$020d	Loose item unavailable foreground
SP.INFWINBD	$020e	Information window border
SP.INFWINBG	$020f	Information window background
SP.INFWINFG	$0210	Information window foreground
SP.INFWINMG	$0211	Information window middleground
SP.SUBINFBD	$0212	Subsidiary information window border
SP.SUBINFBG	$0213	Subsidiary information window background
SP.SUBINFFG	$0214	Subsidiary information window foreground
SP.SUBINFMG	$0215	Subsidiary information window middleground
SP.APPBD	$0216	Application window border
SP.APPBG	$0217	Application window background
SP.APPFG	$0218	Application window foreground
SP.APPMG	$0219	Application window middleground
SP.APPIHIGH	$021a	Application window item highlight
SP.APPIAVABG	$021b	Application window item available background

SP.APPIAVAFG	$021c	Application window item available foreground
SP.APPISELBG	$021d	Application window item selected background
SP.APPISELFG	$021e	Application window item selected foreground
SP.APPIUNABG	$021f	Application window item unavailable background
SP.APPIUNAFG	$0220	Application window item unavailable foreground
SP.SCRBAR	$0221	Pan/scroll bar
SP.SCRBARSEC	$0222	Pan/scroll bar Section (the Section not covered by the bar)
SP.SCRBARARR	$0223	Pan/scroll bar arrow
SP.BUTHIGH	$0224	Button highlight
SP.BUTBD	$0225	Button border
SP.BUTBG	$0226	Button background
SP.BUTFG	$0227	Button foreground
SP.HINTBD	$0228	Hint border
SP.HINTBG	$0229	Hint background
SP.HINTFG	$022a	Hint foreground
SP.HINTMG	$022b	Hint middleground
SP.ERRBG	$022c	Error message background
SP.ERRFG	$022d	Error message foreground
SP.ERRMG	$022e	Error message middleground
SP.SHADED	$022f	Shaded area
SP.3DDARK	$0230	Dark 3D border shade
SP.3DLIGHT	$0231	Light 3D border shade
SP.VERTFILL	$0232	Vertical area fill
SP.SUBTITBG	$0233	Subtitle background
SP.SUBTITTXTBG	$0234	Subtitle text background
SP.SUBTITFG	$0235	Subtitle foreground
SP.MINDEXBG	$0236	Menu index background
SP.MINDEXFG	$0237	Menu index foreground
SP.SEPARATOR	$0238	Seperator lines etc.

Some sort of design guide to help deciding what colour to use (or what some colour is supposed to mean anyway) will hopefully be written at a later stage.

5.3.8.4. New Basic Keywords

There are a number of keywords for palette and colour handling:

5.3.8.4.1. Colours

The first of these are useful for colour handling. Their parameters are exactly the same as for the "normal" commands. The same is true with their names, except for the 'WM_" prefix:

WM_PAPER *[#channel],colour*

Sets the colour which is a word as described above. It also sets the strip as is the case with the normal PAPER command. But there is also the **WM_STRIP** *[#channel],colour* command to set the strip only. Further commands are:

WM_INK *[#channel],colour*
WM_BORDER *[#channel],width,colour*
WM_BLOCK *[#channel],xs,ys,xo,yo,colour*

5.3.8.4.2. Palette handling

There are commands to set/get the system palette and commands to set/get the per job palettes.

5.3.8.4.2.1. System palette keywords

SP_RESET *[#channel] [,number]*

This resets the colour palette given in number to the original values (as configured). Default is number 0.

result% = **SP_GETCOUNT()**

Gets the number of elements contained in a system palette. Each system palette, of course, has the same number of elements.

SP_GET *[number,] address, first, count*

This gets the colours from a system palette and puts them somewhere. The optional "number" parameter tells us which system palette we want (0 to 3, default = 0). "address" is the address of the space for the information, "first" is the number of the first system palette colour to get (starting from 0) and "count" is the number of colours to get.

The space pointed to by "address" MUST have enough space for the number of colours! This is NOT checked by the keyword and it is the programmer's responsibility to make sure that this is so.

As an example, you could use the following code to get ALL of the colours of a system palette:

```
REMark Get number of colours in system palette
totcol%= SP_GETCOUNT
:
REMark enough space for colours + security first=0
address= ALCHP(totcol%*2)+4
SP_GET #1,0,address,first,totcol%
```

SP_SET *[#channel,] [number,] address, first, count*

Sets the system palette entries, the address pointing to a space containing the colours. The parameters are similar to those for SP_GET.

5.3.8.4.2.2. Job palette keywords

SP_JOBPAL *[#channel], Job ID / Job_name, number*

Set the system palette for the job given to the number. The job is given either as a string (e.g. "FiFi") or as a standard Job ID number.

SP_JOBOWNPAL *[#channel], Job ID / Job_name, pal_pointer*

Set the job palette to the palette given in pal_pointer. Of course, the palette must have the format of a standard system palette.

5.3.8.5. New Move modes

As of SMSQ/E v.3.01, new ways of moving a window about the screen have been added. Again, this is a WMAN function but it was thought useful to add it here.

5.3.8.5.1. The move modes

There are now four ways for a window be moved:

0 - **The old way**: the pointer changes to the "move window" sprite which is moved about the screen.

1 - **"Outline"**: click on the move icon with the MOUSE - KEEP HOLDING THE BUTTON DOWN, an outline of the window appears which you can move around and position where you want it. Release the mouse button and the window positions itself correctly.

Please note that you cannot use this move mode with anything but the mouse - the keyboard (cursor keys) will not work.

2 - **"Full window"**. This is the same as 1 above, but instead of an outline, the entire window is moved. For Q40/Q60 users, switching on the cache is advisable...

Please note that you cannot use this move mode with anything but the mouse - the keyboard (cursor keys) will not work.

3 - **"Full window with transparency"** (implemented in SMSQ/E v. 3.16). This is the same as 2 above, but the window to be moved is made "transparent" : one can "see through" it. This is done via "alpha blending". Alpha blending requires A LOT of computing power. So, even if your machine can theoretically handle this type of move, in practice it might not be feasible. For Q40/Q60 users, switching on the Cache is advisable…

This type of move is only implemented for display modes where alpha blending actually makes sense, i.e. modes 16, 32 and 33. In other display modes, such as the QL screen modes, or Atari mono modes, this will be redirected to move mode 2.

Please note that you cannot use this move mode with anything but the mouse - the keyboard (cursor keys) will not work.

5.3.8.5.2. Configuring/setting the move mode

The move modes are configured on a system-wide basis - you cannot have one job moving in mode 0 and the other in mode 1.

Thus, all jobs are affected by the move mode, even those written a long time ago (unless, such as QLiberator, the job doesn't use the WMAN move routine).

The move mode can be changed in two ways:

1 - Configure SMSQ/E (WMAN) to a mode of your liking.

2 - Use the new **WM_MOVEMODE** keyword

This takes one parameter, an integer from 0 to 3:

WM_MOVEMODE 0 : the old way

WM_MOVEMODE 1 : the "outline" move

WM_MOVEMODE 2 : the "full window" move

WM_MOVEMODE 3 : the "full window with transparency" move

5.3.8.5.3. Configuring/setting the degree of transparency

You can set how transparent the window is supposed to be when being moved, from nearly totally transparent to totally opaque. This is done by setting the "alpha value", from 1 (nearly transparent) to 255 (totally opaque).

The alpha value is configured on a system-wide basis - you cannot have one job moving with an alpha value of 100 and the other with 200. Thus, all jobs are affected by this, even those written a long time ago (unless, such as QLiberator, the job doesn't use the WMAN move routine).

The alpha value can be changed in two ways:

1. Configure SMSQ/E (WMAN) to a value of your liking.
2. Use the new **WM_MOVEALPHA** keyword

WM_MOVEALPHA : this new keyword defines the amount of transparency the window should have when moved about, from 1 (nearly transparent) to 255 (totally opaque).

Please note that

1) no check is made on the value passed to this keyword, but only the lower byte is used.

2) a value of 255 is actually equivalent to move mode 2.

3) a value of 0 is allowed but, since this would make the window to be moved totally transparent when it is moved (i.e. you would only ever see the background) this is considered to be an error and a value of 255 will be used!

4) Moving with alpha blending requires a lot of computing power - it may be too slow on your machine.

5.3.8.6. Graphics with alpha blending

(Introduced in SMSQ/E version 3.26)

All graphics operations to the screen, including printing text, can be done with alpha blending, where the object to be drawn, including a single pixel or a text, will be blended in with the background.

To achieve this, set the alpha weight of a channel. This determines how much the object to be displayed is blended into the existing background.

An alpha weight of 0 means that the object to be drawn will be practically totally translucent, i.e. it can't be seen since it lets the background shine through entirely. An alpha weight of 255 means that the object to be displayed is totally opaque, i.e. it covers the existing background.

There is a program called "dev8_extras_alpha_test_bas" in the SMSQ/E sources which can show you how this works (in SBasic).

IMPORTANT NOTE: This call only really makes sense for 16 bit modes. In 8 bit Aurora mode, the trap tries as well as it can, but don't expect miracles, there just are not enough colours. When in QL modes 4 or 8, or in Atari monochrome mode, there simply is no alpha blending (note that being in QL modes 4 or 8 is not the same as having used COLOUR_QL in 16 or 8 bit mode).

5.3.8.6.1. Machine code interface

Trap #3 D0=$62 **IOW.SALP**

Set the alpha blending weight for window

Call parameters		**Return parameters**	
D1.B	alpha weight (0..255)	D1	Preserved
D2.		D2	Preserved
D3.W	Timeout	D3.L	Preserved
		D4+	All preserved
A0	Channel ID	A0	Preserved
A1		A1	Preserved
A2		A2	Preserved
A3		A3	Preserved
		A4+	All preserved

Error returns:

ICHN channel not open

This call affects all following text and graphics output functions. To disable alpha blending set the weight to 255.

5.3.8.6.2. S*Basic keywords

Use the **ALPHA_BLEND** command:

ALPHA_BLEND *[#channel,] weight*.

Sets the alpha weight for this channel. All further output to this channel will use this. To switch it off, set weight to 255.

- Channel is the channel to which this applies, as usual, it defaults to 1.
- Weight is the alpha weight of future display operations: from 0 (translucent) to 255 (opaque).

6. QDOS Device Drivers

A user-supplied QDOS device driver is a collection of routines which allow an application 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.

Both drivers and tasks are linked in to lists provided by the operating system.

The following traps are used to add 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

The following traps are used to remove items from those lists:

SMS.REXI	unlinks in an external interrupt service task
SMS.RPOL	unlinks in a 50/60 Hz polling service task
SMS.RSHD	unlinks in a scheduler loop task
SMS.RIOD	unlinks in a device driver to the I/O system
SMS.RFSD	unlinks in a directory device driver to the file system

The operating system 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 memory 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 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 definition 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	pointer to device name	A0	channel definition block
		A1-A2	???
A3	base of device driver definition block	A3	???
A4-A5		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		D1-D3	???
		D4-D7	All preserved
A0	pointer to base of channel definition block	A0	???
		A1-A2	???
A3	pointer to base of device driver definition 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 assumed that CLOSE cannot fail. Registers D4 to D7 and A4 to A6 must be set to their initial values before return.

6.4.3. 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	pointer to base of channel definition block	A0	preserved
A1	additional information	A1	updated parameter
A2	additional information	A2	preserved
A3	pointer to base of device driver definition 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. 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 and level 3 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)
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 18.8 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 definition 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 D0.

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 definition
      MOVE.W      MEM.RCHP,A4       routine to release channel definition 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 to 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 definition 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
```

7.4. 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	pointer to medium name	A1	???
A2		A2	???
A3	base of device driver definition block	A3	???
		A4-A5	???
A6	system variables	A6	preserved

Error returns:

FMTF format failed

8. 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 FAT). The first block of the map will be in sector 1 side 0 track 0. Note that in QL parlance, a cluster is called a group.

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). **Note**: this is not a standard QL string as there is no length word
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 (normally <=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 allocation group)
Q5A_EODR	$22	long	current end of directory (block/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		Max length

The map is always of a size to fill the first three (logical) sectors of the drive, being padded with 'non-existent' sectors if necessary to fill the (512*3-96)/3=480 allocations 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 existent 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 S*Basic, 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 I/O 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 S*Basic using the normal string I/O 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] [SMSQ/E]

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 Hard disk or Changeable Disk file
WIN1 refers to Hard disk "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. Interfacing to S*Basic

When writing S*Basic 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 S*Basic variables and arrays, he may wish to access or modify the S*Basic 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 S*Basic Area

The S*Basic area contains twelve distinct areas:

the job header,
the S*Basic 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 S*Basic area, and looks just like other job header (see Section 18.5). Immediately above this is the S*Basic 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 S*Basic 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 S*Basic 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 S*Basic 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 S*Basic variables and parameters. The channel table is required in order to access S*Basic 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
pointer to RI stack	-1	$0101	String expression
pointer to RI stack	-1	$0102	Floating point expression
pointer 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	S*Basic procedure
Line no in msw	Name pointer	$0501	S*Basic string function
Line no in msw	Name pointer	$0502	S*Basic f.p. function
Line no in msw	Name pointer	$0503	S*Basic integer function
Value pointer	Name pointer	$0602	REPeat loop index – floating point
Value pointer	Name pointer	$0603	REPeat loop index - integer
Value pointer	Name pointer	$0702	FOR loop index – floating point
Value pointer	Name pointer	$0703	FOR loop index - integer
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 S*Basic are typed according to whether the name ends in %, $ or neither. Functions written in machine code, in common with procedures written in S*Basic 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 normalised 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	"" (empty string)

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 simple set of rules for writing procedures in machine code for S*Basic:

1. As the S*Basic 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. This is not true for SMSQ/E.

   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 S*Basic procedures and functions written in machine code may be linked into the name table using the vectored routine **SB.INIPR** (see Section 16). 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 S*Basic 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 S*Basic, 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 S*Basic 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 S*Basic 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 S*Basic 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 S*Basic area and the S*Basic 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 S*Basic 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 occurred, 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 S*Basic 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. 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 Kb 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 Kb)

$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 or other machines running SMSQ/E, the area from $C0000 is filled up with continuous memory (up to $3FFFFF or beyond).

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. It notably is different on machines running SMSQ/E in higher display modes.

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 grey-scale.

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.

In 8 bit (aurora or palette) modes, there is one byte per pixel. In 16 bit modes, there are two bytes per pixel.

The 16 bit QPC/QXL/SMSQmulator format is as follows:

G2 G1 G0 B4 B3 B2 B1 B0 R4 R5 R2 R1 R0 G5 G4 G3

The 16 bit Q40/Q60 format is as follows:

G4 G3 G2 G1 G0 R4 R3 R2 R1 R0 B4 B3 B2 B1 B0 W

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 8049single-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 de-bounce 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 S*Basic 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 example shows how a microdrive is selected or de-selected:

```
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    pointer 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
```

10.9. 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 exception 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.

10.10. 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 yours. 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.

10.11. 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.

10.12. Different Processors [ST][SMSQ/E]

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 program 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.

10.13. 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.
17 for the Q40/Q60.
20 for SMSQmulator.
24 for the TT.
28 for the QXL.
30 for QPC.

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

The display types are:

%00000000	Standard QL or the Futura emulator (we cannot tell whether it gives real MODE8 or not)
%01000000	The Extended 4 Emulator.
%10000000	The QVME emulator card.
%00100000	ATARI monochrome mode.
%11000000	VGA mode (e.g. QXL) or QL mode LCD.
%10100000	Aurora.

10.14. 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 SLM605).

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. 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 card 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 startup 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 four 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 S*Basic.

The list of BASIC procedures and functions is in the form used by **SB.INIPR** (see Section 16).

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. 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 S*Basic **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. 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].

* The number of bytes allocated as returned in D1 includes the bytes necessary for the heap header. It does not correspond to the number of bytes that may be used in the heap (which is smaller than the number returned in D1).

Trap #1 D0=$0A

SMS.ACJB

Activate 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 trap will then return with the error code (if any) from that job.

Trap #1 D0=$0C

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	pointer to pointer 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 4.1 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	Number. 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=$0E

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 allocate (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

SMS.CACH [SMSQ]

Turn Cache on or off

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

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	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=$01 **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.

In QDOS and in versions of SMSQ/E before 3.24, care should be taken that the parameters passed in D2 and D3 are both even before calling this trap. If they are not, the resulting job will most likely crash.

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.

Obviously, 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=$07 **SMS.EVX**

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

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	**SMS.FPRM** [SMSQ]
		Find Preferred Module
Trap #1	D0=$31	**SMS.LENQ** [SMSQ]
		Language Enquiry
Trap #1	D0=$30	**SMS.LLDM** [SMSQ]
		Link in Language Dependent Module
Trap #1	D0=$32	**SMS.LSET** [SMSQ]
		Language Set
Trap #1	D0=$34	**SMS.MPTR** [SMSQ]
		Find Message Pointer
Trap #1	D0=$33	**SMS.PSET** [SMSQ]
		Set Printer Translate

For details on these trap calls, please refer to Section 19 "Language handling in SMSQ".

Trap #1 D0=$05 **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=$0A).

Trap #1 D0=$06

SMS.FRTP

Find largest contiguous free space that may be allocated in transient program 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=$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:

$09	Read a row of the keyboard, 1 parameter		
	4 bits	The row number	
	8 bits	Reply	
$0A	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		
$0B	Kill sound, no parameters, no reply.		

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

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

This is equivalent to the S*Basic command:

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

Trap #1 D0=$00

SMS.INFO

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 Variables
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. This trap always succeeds.

Trap #1 D0=$02

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 **SMS.IOPR [SMSQ]**

Set IO Priority

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

Error returns

Always okay

The I/O priority sets the priority of the I/O retry operations.

In effect, this sets a limit on the time spent by the scheduler retrying I/O operations.

A priority of one sets the I/O 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=$1A	Link an external interrupt service routine	**SMS.LEXI**
	D0=$1C	Link a polling 50/60 Hz service routine	**SMS.LPOL**
	D0=$1E	Link a scheduler loop task	**SMS.LSHD**
	D0=$20	Link an IO device driver	**SMS.LIOD**
	D0=$22	Link a directory device driver into the operating system	**SMS.LFSD**

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=$19

SMS.RCHP

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

Please refer to Section 2.1.4 for an explanation of the common heap.

Trap #1 D0=$0D

SMS.REHP

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	Pointer to Pointer 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	Remove an external interrupt service routine	**SMS.REXI**
	D0=$1D	Remove a polling 50/60 Hz service routine	**SMS.RPOL**
	D0=$1F	Remove a scheduler loop task	**SMS.RFSD**
	D0=$21	Remove an IO device driver	**SMS.RIOD**
	D0=$23	Remove a directory device driver from the operating system	**SMS.RFSD**

Call parameters		**Return parameters**	
D1		D1	Preserved
D2		D2	Preserved
D3		D3+	All preserved
A0	Address of link	A0	Preserved
A1		A1	???
A2		A2	Preserved
A3		A3	Preserved

Trap #1 D0=$04 **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	Number of bytes released
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=$38

SMS.SCHP [SMSQ]

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=$0B

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

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=$08

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 timeout period is specified as -1, then the suspension is indefinite; no other negative value should be used.

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 job is already suspended, the suspension will be reset. All jobs are rescheduled.

Trap #1 D0=$24

SMS.TRANS [not SMS2]

Set translation table and error messages

Call parameters		Return parameters	
D1	Pointer to translation table, -1 or 0 (or 1)	D1	???
D2.L	Pointer 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=$09

SMS.USJB

Release a job

Call parameters		Return parameters	
D1.L	Job ID	D1.L	Job ID
D2		D2	Preserved
D3		D3+	All preserved
A0		A0	Base of job control area
A1		A1	Preserved
A2		A2	Preserved
A3		A3+	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=$3B **SMS.WEVT [SMSQ]**

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 **SMS.XTOP [SMSQ]**

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.

CRJB

13.1. Trap 1 Keys - numerical order with page reference

(Clicking on the page number will send you there)

SMS.INFO	$00	get INFOrmation on SMS	12
SMS.CRJB	$01	CReate Job	6
SMS.INJB	$02	get INformation on JoB	12
SMS.RMJB	$04	ReMove JoB	15
SMS.FRJB	$05	Forced Remove JoB	9
SMS.FRTP	$06	find largest FRee space in Tpa	10
SMS.EXV	$07	set EXception Vector	8
SMS.SSJB	$08	SuSpend a JoB	19
SMS.USJB	$09	UnSuspend a JoB	20
SMS.ACJB	$0a	ACtivate a JoB	2
SMS.SPJB	$0b	Set Priority of JoB	18
SMS.ALHP	$0c	ALlocate in HeaP	3
SMS.REHP	$0d	RElease to HeaP	14
SMS.ARPA	$0e	Allocate in Resident Procedure Area	4
SMS.DMOD	$10	set or read the Display MODe	4
SMS.HDOP	$11	do a Hardware Dependent Operation	10
SMS.COMM	$12	set COMMuncation baud rate etc.	5
SMS.RRTC	$13	Read Real Time Clock	16
SMS.SRTC	$14	Set Real Time Clock	19
SMS.ARTC	$15	Adjust Real Time Clock	4
SMS.AMPA	$16	Allocate space in S*Basic area	3
SMS.RMPA	$17	Release space in S*Basic area	16
SMS.ACHP	$18	Allocate space in Common HeaP	1
SMS.RCHP	$19	Release space in Common HeaP	14
SMS.LEXI	$1a	Link in EXternal Interrupt action	13
SMS.REXI	$1b	Remove EXternal Interrupt action	15
SMS.LPOL	$1c	Link in POLled action	13
SMS.RPOL	$1d	Remove POLled action	15
SMS.LSHD	$1e	Link in ScHeDuler action	13
SMS.RSHD	$1f	Remove ScHeDuler action	15
SMS.LIOD	$20	Link in IO Device driver	13
SMS.RIOD	$21	Remove IO Device driver	15
SMS.LFSD	$22	Link in Filing System Device driver	13
SMS.RFSD	$23	Remove Filing System Device driver	15
SMS.TRNS	$24	Set translation and error messages	20
SMS.XTOP	$25	eXTernal Operation [SMSQ]	21
SMS.IOPR	$2e	IO PRiority [SMSQ]	13
SMS.CACH	$2f	CACHe handling [SMSQ]	5
SMS.LLDM	$30	Link in Language Dependent Module [SMSQ]	9 / Section 19

SMS.LENQ	$31	Language ENQuiry [SMSQ]	9 / Section 19
SMS.LSET	$32	Language SET [SMSQ]	9 / Section 19
SMS.PSET	$33	Printer translate SET [SMSQ]	9 / Section 19
SMS.MPTR	$34	find a Message PoinTeR [SMSQ]	9 / Section 19
SMS.FPRM	$35	Find PReferred Module [SMSQ]	9 / Section 19
SMS.SCHP	$38	Shrink alloaction in common heap [SMSQ]	17
SMS.SEVT	$3a	Send event to job [SMSQ]	18
SMS.WEVT	$3b	Wait for event [SMSQ]	21

14. I/O Management Traps

Trap #2 D0=$02 **IOA.CLOS**

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

Trap #2 D0=$06 **IOA.CNAM [SMSQ]**

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	Pointer to buffer	A1	Device name (QDOS-string)
A2		A2	Preserved
		A3+	All preserved

Error returns:

ICHN Channel not open

IPAR Buffer too small

Trap #2 D0=$04

IOA.DELF

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.

NOTE: not all device drivers/OSes support this TRAP#2 call. In this case, an IOA.OPEN call with D3 set to -1 should be used.

SMSQ/E checks whether a device driver is specially marked as being compatible with the IOA.DELF TRAP#2 call, if not it will use the IOA.OPEN call with D3 set to -1. DV3 drivers linked in via the standard *dv3_link* subroutine and a standard table of values, will be marked as being compatible with the IOA.DELF TRAP#2 call, and the deletion will be handled internally by SMSQ/E, without the device driver specifically having to take care of this.

Trap #2 D0=$03

IOA.FRMT

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	Pointer 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=$01

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		
	-1 Delete file		
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 on all versions of QDOS.

Note also that most device drivers, when requested to open a directory will, if no such directory exists, open the next existing higher level directory. Most QL software expects this behaviour.

Trap #2 D0=$05

IOA.SOWN [SMSQ]

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

14.1. Trap 2 Keys - numerical order with page reference

IOA.OPEN	$01	OPEN IOSS channel	4
IOA.CLOS	$02	CLOSe IOSS channel	1
IOA.FRMT	$03	FoRMaT medium on device	3
IOA.DELF	$04	DELete file from device	2
IOA.SOWN	$05	Set OWNer of channel [SMSQ]	5
IOA.CNAM	$06	fetch Channel NAMe [SMSQ]	1

15. 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.

Trap #3 D0=$04 **IOB.ELIN**

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

This is 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.

Trap #3 D0=$01

IOB.FBYT

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

Trap #3 D0=$02 Fetch a line of characters terminated by ASCII <LF> * **IOB.FLIN**

D0=$03 Fetch a string of bytes **IOB.FMUL**

Call parameters		**Return parameters**	
D1		D1.W	Number of bytes fetched
D2.W	Length of buffer (+ve word)	D2	Preserved but upper word 0
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

Error returns:

NC	Not complete
ICHN	Channel not open
EOF	End of file
BFFL	Buffer full

The character count of a fetch a line trap includes the linefeed character ASCII $0A, if found. The length in D2 must be a positive word (i.e. no more that $7FFF). **NOTE :** if no LF can be found within the length of the buffer, the trap should return error **ERR.BFFL** (not OVFL as previously mentioned). Whilst the number of bytes fetched is returned in D1.W, the upper word of D1 is NOT preserved.

*** Note** : For IOB.FLIN, many SMSQ/E level 3 device drivers (mostly WIN and FLP) will convert a <CR><LF> to a simple <LF>.

Trap #3 D0=$05 **IOB.SBYT**

Send a byte

Call parameters		**Return parameters**	
D1.B	Byte to be sent	D1	???
D2		D2	Preserved
D3.W	Timeout	D3 +	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.

Trap #3 D0=$07 **IOB.SMUL**

Send a string of bytes

Call parameters		**Return parameters**	
D1		D1.W	Number of bytes sent
D2.W	Number of bytes to be sent (+ve word)	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. **Note**: the size of the number of bytes to be sent in D2 should be a positive word, i.e. no larger than $7FFF. Whilst the number of bytes sent is returned in D1.W, the upper word of D1 is NOT preserved.

Trap #3 D0=$06

IOB.SUML [SMSQ/E]

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	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 ($07) 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 safe way of sending graphics data or control codes to the printer, as they will never be translated into other byte patterns.

This trap is only available on SMSQ/E.

Trap #3 D0=$00

IOB.TEST

Check for pending input

Call parameters		Return parameters	
D1		D1	???
D2		D2	Preserved
D3.W	Timeout	D3.L	Preserved
		D4+	
A0	Channel ID	A0	Preserved
A1		A1	???
A2		A2	Preserved
A3		A3+	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.

Trap #3 D0=$40

IOB.CHEK

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+	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

IOF.DATE[EXT][DD2]

Set or read file date

Call parameters			Return parameters	
D1.L	-1	Read date	D1.L	Date set
	0	Set date		Date read
	Date			
D2.B	0	Update date	D2	Preserved
	2	Backup date		
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 or 3 devices.

Trap #3 D0=$41

IOF.FLSH

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.

Trap #3 D0=$48

IOF.LOAD

Load a file into memory

Call parameters		Return parameters	
D1		D1	???
D2.L	Length of file	D2	Preserved
D3.W	Timeout (should be -1)	D3+	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.

Trap #3 D0=$45

IOF.MINF

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	Pointer to 10 byte buffer	A1	
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

IOF.MKDR [DD2]

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+	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+	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.

Trap #3 D0=$47 **IOF.RHDR**

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+	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.

Trap #3 D0=$4A **IOF.RNAM [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.

Trap #3 D0=$49

IOF.SAVE

Save an entire file

Call parameters		Return parameters	
D1		D1	???
D2.	Length of file	D2	Preserved
D3.W	Timeout (should be -1)	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.

Trap #3 D0=$46

IOF.SHDR

Set file header

Call parameters		Return parameters	
D1		D1.W	Length of header set
D2		D2	Preserved
D3.W	Timeout	D3.	Preserved
		D4+	All preserved
A0	Channel ID	A0	Preserved
A1	Base of header definition	A1	End of header definition
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 file 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

IOF.TRNC [EXT][DD2]

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

IOF.VERS [DD2]

Set or read file version

Call parameters		Return parameters	
D1.L	Read: -1	D1.L	File version
	Set: 0		
	Version: 1 to 65535		
D2		D2	Preserved
D3.W	Timeout	D3	Preserved
A0	Channel ID	A0	Preserved
A1		A1	Preserved
A3		A3	Preserved
		A4+	All 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 and 3 devices only.

Trap #3 D0=$4F

IOF.XINF [DD2]

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.

The default is: height of window = 100 units.

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

Trap #3 D0=$35

IOG.FILL

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**
	D0=$5C	(8 bit palette)	**IOW.BLKP** [SMSQ/E]
	D0=$5D	(24 bit)	**IOW.BLKT** [SMSQ/E]
	D0=$5E	(native)	**IOW.BLKN** [SMSQ/E]

Fill rectangular block in window

Call parameters		**Return parameters**	
D1	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+	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.

The colour to be set is in D1, the actual amount used depends on the mode : a byte for **iow.blok** and in palette mode, the lower 3 bytes in 24 bit mode and a variable amount in native mode.

Note: **The last three traps are only available under SMSQ/E.**

Trap #3 D0=$0B

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+	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=$0F

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+	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=$0C

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 +	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).

Trap #3 D0=$0D

IOW.DEFW

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+	All preserved

Error returns:

NC	Not complete
ICHN	Channel not open
ORNG	Window does not fit on screen

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.

Trap #3 D0=$2E

IOW.DONL

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=$0E

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.

Trap #3 D0=$25

IOW.FONT

Set or reset the fount

Call parameters		Return parameters	
D1		D1	???
D2	0 (or "DEFF" [SMSQ/E])	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.

[SMSQ/E] In **SMSQ**, 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. Moreover, 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=$60	Define QL colour palette	**IOW.PALQ** [SMSQE]
	D0=$61	Define 8-bit colour palette	**IOW.PALT** [SMSQE]

Call parameters		**Return parameters**	
D1.W	Start entry in palette	D1	???
D2.W	Number of entries to change	D2	???
D3.W	Timeout	D3.L	Preserved
		D4+	All preserved
A0	Channel ID	A0	Preserved
A1	Pointer to entries	A1	???
A2		A2	Preserved
A3		A3	Preserved
		A4+	All preserved

Error returns:

NC	Not complete
ICHN	Channel not open

These traps redefine colour palettes, either the QL palette or the 8 bit palette.

On entry to the traps, D1 is the number of the first entry to change (starting at 0 for the first entry in the palette). D2 is the number of palette entries to change from there on.

There are 8 entries in the QL palette and 256 entries in the 8 bit palette.

A1 points to the list of new palette entries. Each entry in the list takes one long word, which must be in 24 bit RGBx format (i.e. the colour information is in the 24 MSb of the long word – the lower byte is ignored).

Please note that these traps are only available under SMSQ/E.

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=$50	Set paper colour (palette)	**IOW.PAPP** [SMSQE]
	D0=$51	Set strip colour (palette)	**IOW.STRP** [SMSQE]
	D0=$52	Set ink colour (palette)	**IOW.INKP** [SMSQE]
	D0=$53	Set border colour (palette)	**IOW.BORP** [SMSQE]
	D0=$54	Set paper colour (24 bit)	**IOW.PAPT** [SMSQE]
	D0=$55	Set strip colour (24 bit)	**IOW.STRT** [SMSQE]
	D0=$56	Set ink colour (24 bit)	**IOW.INKT** [SMSQE]
	D0=$57	Set border colour (24 bit)	**IOW.BORT** [SMSQE]
	D0=$59	Set paper colour (native)	**IOW.PAPN** [SMSQE]
	D0=$59	Set strip colour (native)	**IOW.STRN** [SMSQE]
	D0=$5A	Set ink colour (native)	**IOW.INKN** [SMSQE]
	D0=$5B	Set border colour (native)	**IOW.BORN** [SMSQE]

Call parameters		**Return parameters**	
D1	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

These traps set the paper, ink, strip and border colours for the respective modes. The colour to be set is in D1, the actual size (amount used) depends on the mode : a byte in palette mode, the lower 3 bytes in 24 bit mode and a variable amount in native mode.

Please note that these traps are only available under SMSQ/E.

Trap #3 D0=$0A

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.

Trap #3 D0=$26

IOW.RCLR

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+	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=$62

IOW.SALP

Set the alpha blending weight for window

Call parameters		Return parameters	
D1.B	alpha weight (0..255)	D1	Preserved
D2.		D2	Preserved
D3.W	Timeout	D3.L	Preserved
		D4+	All preserved
A0	Channel ID	A0	Preserved
A1		A1+	All Preserved

Error returns:

ICHN	channel not open

This call affects all following text and graphics output functions. To disable alpha blending set the weight to 255.

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 next column	**IOW.PCOL**
	D0=$14	Move cursor to previous row	**IOW.NCOL**
	D0=$15	Clear right hand end of cursor line	**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

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.

Trap #3 D0=$2A Set flash attribute

D0=$2B Set underline attribute

IOW.SFLA
IOW.SULA

Call parameters

D1.B 0=attribute off, else attribute on
D2
D3.W Timeout

A0 Channel ID
A1
A2

Return parameters

D1 ???
D2.L Preserved
D3.L Preserved
D4+ All preserved

A0 Preserved
A1 ???
A2+ All preserved

Error returns:

NC Not complete
ICHN Channel not open

Trap #3 D0=$2C

IOW.SOVA

Set the character writing or plotting mode

Call parameters

D1.W Mode:
- -1 Ink is exclusive ORed into the background
- 0 Character background is strip colour
- 1 Character background is transparent

D2
D3.W Timeout

A0 Channel ID
A1
A2
A3

Return parameters

D1 ???
D2.L Preserved
D3.L Preserved
D4+ All preserved

A0 Preserved
A1 ???
A2 Preserved
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

IOW.SPIX

Set cursor to pixel position

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
A3		A3	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 **IOW.SSIZ**

Set character size and spacing

Call parameters		**Return parameters**	
D1.W	Character width/spacing:	D1	???
	0 Single width, 6 pixel spacing		
	1 Single width, 8 pixel spacing		
	2 Double width, 12 pixel spacing		
	3 Double width, 16 pixel spacing		
D2	Character height / spacing:	D2.L	Preserved
	0 Single height, 10 pixel spacing		
	1 Double height, 20 pixel spacing		
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.

Trap #3 D0=$09

IOW.XTOP

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.

15.1. Trap 3 Keys - numerical order with page reference

IOB.TEST	$00	TEST input	6
IOB.FBYT	$01	Fetch BYTe from input	2
IOB.FLIN	$02	Fetch LINe from input	3
IOB.FMUL	$03	Fetch MULtiple characters/bytes	3
IOB.ELIN	$04	Edit LINe of characters	1
IOB.SBYT	$05	Send BYTe to output	3
IOB.SUML	$06	Send a string of untranslated bytes [SMSQ/E]	5
IOB.SMUL	$07	Send MULtiple bytes	4
IOW.XTOP	$09	eXTernal OPeration on screen	37
IOW.PIXQ	$0A	PIXel coordinate Query	29
IOW.CHRQ	$0B	CHaRacter coordinate Query	20
IOW.DEFB	$0C	DEFine Border	22
IOW.DEFW	$0D	DEFine Window	23
IOW.ECUR	$0E	Enable CURsor	24
IOW.DCUR	$0F	Disable CURsor	22
IOW.SCUR	$10	Set CURsor position (character coordinates)	32
IOW.SCOL	$11	Set cursor COLumn	32
IOW.NEWL	$12	put cursor on a NEW Line	32
IOW.PCOL	$13	move cursor to Previous COLumn	32
IOW.NCOL	$14	move cursor to Next COLumn	32
IOW.PROW	$15	move cursor to Prevous ROW	32
IOW.NROW	$16	move cursor to Next ROW	32
IOW.SPIX	$17	Set cursor to PIXel position	35
IOW.SCRA	$18	SCRoll All of window	31
IOW.SCRT	$19	SCRoll Top of window (above cursor)	31
IOW.SCRB	$1A	SCRoll Bottom of window (below cursor)	31
IOW.PANA	$1B	PAN All of window	27
IOW.PANL	$1E	PAN cursor Line	27
IOW.PANR	$1F	PAN Right hand end of cursor line	27
IOW.CLRA	$20	CLeaR All of window	21
IOW.CLRT	$21	CLeaR Top of window (above cursor)	21
IOW.CLRB	$22	CLeaR Bottom of window (below cursor)	21
IOW.CLRL	$23	CLeaR cursor Line	21
IOW.CLRR	$24	CLeaR Right hand side of cursor line	21
IOW.FONT	$25	set / read FOuNT (font U.S.A.)	25
IOW.RCLR	$26	ReCoLouR a window	30
IOW.SPAP	$27	Set PAPer colour	34
IOW.SSTR	$28	Set STRip colour	34
IOW.SINK	$29	Set INK colour	34
IOW.SFLA	$2A	Set FLash Attribute	33

IOW.SULA	$2B	Set UnderLine Attribute	33
IOW.SOVA	$2C	Set OVerwrite Attributes	33
IOW.SSIZ	$2D	Set character SIZe	36
IOW.BLOK	$2E	fill a BLOcK with colour	19
IOW.DONL	$2F	DO a pending NewLine	23
IOG.DOT	$30	draw (list of) DOTs	16
IOG.LINE	$31	draw (list of) LINEs	16
IOG.ARC	$32	draw (list of) ARCs	16
IOG.ELIP	$33	draw ELlIPse	16
IOG.SCAL	$34	set graphics SCALe	16
IOG.FILL	$35	set area FILL	18
IOG.SGCR	$36	Set Graphics CuRsor position	16
IOF.CHEK	$40	CHEcK all pending operations on file	6
IOF.FLSH	$41	FLuSH all buffers	8
IOF.POSA	$42	set file POSition to Absolute address	11
IOF.POSR	$43	move file POSition Relative to current position	11
IOF.MINF	$45	get Medium INFormation	9
IOF.SHDR	$46	Set file HeaDeR	13
IOF.RHDR	$47	Read file HeaDeR	12
IOF.LOAD	$48	(scatter) LOAD file	9
IOF.SAVE	$49	(scatter) SAVE file	13
IOF.RNAM	$4A	ReNAMe file [EXT, DD2]	12
IOF.TRNC	$4B	TRuNCate file to current position [EXT, DD2]	14
IOF.DATE	$4C	set or get file DATEs [EXT,DD2]	7
IOF.MKDR	$4D	MaKe DiRectory [DD2]	10
IOF.VERS	$4E	set or get VERSion [DD2]	14
IOF.XINF	$4F	get eXtended INFormation [DD2]	15
IOW.PAPP	$50	Set paper colour (palette) [SMSQ/E]	28
IOW.STRP	$51	Set strip colour (palette) [SMSQ/E]	28
IOW.INKP	$52	Set ink colour (palette) [SMSQ/E]	28
IOW.BORP	$53	Set border colour (palette) [SMSQ/E]	28
IOW.PAPT	$54	Set paper colour (24 bit) [SMSQ/E]	28
IOW.STRT	$55	Set strip colour (24 bit) [SMSQ/E]	28
IOW.INKT	$56	Set ink colour (24 bit) [SMSQ/E]	28
IOW.BORT	$57	Set border colour (24 bit) [SMSQ/E]	28
IOW.PAPN	$58	Set paper colour (native) [SMSQ/E]	28
IOW.STRN	$59	Set strip colour (native) [SMSQ/E]	28
IOW.INKN	$5A	Set ink colour (native) [SMSQ/E]	28
IOW.BORN	$5B	Set border colour (native) [SMSQ/E]	28
IOW.BLKP	$5C	Fill block with colour (palette) [SMSQ/E]	19
IOW.BLKT	$5D	Fill block with colour (24 bit) [SMSQ/E]	19
IOW.BLKN	$5E	Fill block with colour (native) [SMSQ/E]	19
IOW.PALQ	$60	Define QL colour palette [SMSQ/E]	26

IOW.PALT	$61	Define 8-bit colour palette [SMSQ/E]	26
IOW.SALP	$62	Set the alpha blending weight for window	30

16. Vectored Routines

Vector	$D6	**CV.DATIL [SMS]**

Convert date and time to Integer Long

Call parameters		**Return parameters**	
D1		D1	Date
D2		D2.L	Preserved
D3		D3.L	Preserved
A0		A0	Preserved
A1	Pointer 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 non-SMSQE QL-Emulator. It is available on all machines that run SMSQ/E.

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.L	???
D3		D3.L	???
D7	0 or pointer to end of buffer	D7	Preserved
A0	Pointer to buffer (rel. A6)	A0	Updated to end of buffer+1
A1	Pointer 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 (long) 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.L	???
D3		D3.L	???
A0	Pointer to buffer (rel. A6)	A0	Pointer to buffer (rel. A6)
A1	Pointer 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.W		D2	Preserved
D3.W		D3	Preserved
A0		A0	Preserved
A1	Pointer 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**.

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.W		D2	Preserved / Free space
D3.W		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.

Vector $122

IOU.DNAM

Decode device name

Call parameters		Return parameters	
D1		D1	???
D2		D2	???
D3		D3	???
		D4+	All preserved
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 (starting at 1), or zero if not . All returned parameters are word length integers.

Examples:

The CON description is:	
DC.W 3,'CON'	Console
DC.W 5	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.W 3,'SER'	Rs232 serial device
DC.W 4	Four parameters
DC.W -1,1	Port number (default 1)
DC.W 4,'OEMS'	Parity (odd/even/mark/space)
DC.W 2,'IH'	Ignore/use handshaking
DC.W 3,'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 SMSQ/E), otherwise D0 is smashed.

Vector	$E8	Direct queue handling	**IOU.SSQ**
	$EA	General I/O handling	**IOU.SSIO**

Call parameters		**Return parameters**	
D1	Standard IOSS value	D1	Standard IOSS value
D2	Standard IOSS value	D2.L	Standard IOSS value
D3	Standard IOSS value	D3.L	???
A0	Standard IOSS value	A0	Preserved
A1	Standard IOSS value	A1	Standard IOSS value
A2		A2	???
A3		A3	???

Error returns:

IPAR	Undefined action
ICHN	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

           MOVE.W      $E8,A4                           DC.L  TEST
           JSR         (A4)                             DC.L  FETCH
           DC.L        TEST                             DC.L  SEND
           DC.L        FETCH                            RTS
           DC.L        SEND
           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	**MD.READ** [QL]
	$126	Write a sector	**MD.WRITE** [QL]
	$128	Verify a sector	**MD.VERIF** [QL]
	$12A	Read a sector header	**MD.RDHDR** [QL]

Call parameters		**Return parameters**	
D1		D1	File number (read/verify)
D2		D2	Block number (read/verify)
D7		D7	Sector number (read header)
A0		A0	???
A1	Pointer to start of buffer	A1	Standard IOSS value
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
```

Vector $C0 Allocate common heap area **MEM.ACHP**

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	??? (unmodified in [SMSQ/E])
A6	pointer 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.

Vector $D8 Allocate an area in a heap **MEM.ALHP**

Call parameters		**Return parameters**	
D1.L	Length required	D1	Length allocated
D2		D2.L	???
D3		D3.L	???
A0	Pointer to pointer to free space	A0	Base of area allocated
A1		A1	???
A2		A2	???
A3		A3	???

Error returns:

IMEM No free space large enough

The condition code is not cleared on success on all ROM versions

See Section 4.1 for details of the heap allocation mechanism. The area allocated is not cleared.

Vector	$D2	Link an item into a list	**MEM.LLST**
	$D4	Unlink an item from a list	**MEM.RLST**

Call parameters		**Return parameters**	
D1		D1	Preserved
D2		D2.L	Preserved
D3		D3.L	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 Release common heap space **MEM.RCHP**

Call parameters		Return parameters	
D1		D1	???
D2		D2.L	???
D3		D3.L	???
A0	Base of area to release	A0	???
A1		A1	???
A2		A2	???
A3		A3	???
A6	Pointer 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**.

Vector $DA Link a free space (back) into a heap **MEM.REHP**

Call parameters		Return parameters	
D1.L	Length to link in	D1	???
D2		D2.L	???
D3		D3.L	???
A0	Base of new space	A0	???
A1	Pointer to pointer to free space	A1	???
A2		A2	???
A3		A3	???

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**	
D		D1	???
D2		D2	???
D3		D3	???
A0	Pointer to name (OPW.WIND only)	A0	channel ID
A1	Pointer to parameter block	A1	???
A2		A2	???
A3		A3	???

Error returns:

INAM	Bad device name (window 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	pointer to RI stack (rel. A6)	A1	Updated
A2		A2	Preserved
A3	Absolute pointer to operation list (QA.MOP)	A3	Preserved
A4	Pointer to base of variables 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	-6
		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	
$09	**qa.fltli**	float a long integer	-2
$0D	**qa.halve**	TOS / 2	0

$0F	**qa.doubl**	TOS * 2	0
$11	**qa.recip**	1 / TOS	0
$13**l**	**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 Reserve Room on Arithmetic Stack **QA.RESRI**

Call parameters		**Return parameters**	
D1.L	Number of bytes required	D1	???
D2		D2.L	???
D3		D3.L	???
A0		A0	Preserved
A1	Pointer to RI stack (rel. A6) [QDOS]	A1	???
A2		A2	Preserved
A3		A3	Preserved

Error returns:

none Nothing useful: the content of D0 on return from this call is spurious, see below.

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

QA.RESRI is used to reserve space on the arithmetic stack..

One should not test the value of D0 on return from this call, the value returned is spurious.

Since not only the stack but the whole S*Basic area may move during the call, the arithmetic stack pointer should be saved in **SB_ARTHP(A6)** (=BV_RIP(A6)), whence it should be retrieved after the call has been completed.

On SMSQ/E it is not necessary for A1 to contain the ARI stack pointer before calling this vector and this call might fail if there is not sufficient memory. In this case, though, the call to this vector will not return to the caller when the error IMEM is generated, but will be diverted to the general SMSQ/E SBasic error handling routines.

NOTE: Under SMSQ/E at least, this call simply does nothing when called from a compiled job.

Vector	$112	S*Basic get Integer parameter(s)	**SB.GTINT**
	$114	S*Basic get Floating point parameter(s)	**SB.GTFP**
	$116	S*Basic get String parameter(s)	**SB.GTSTR**
	$118	S*Basic get Long Integer parameter(s)	**SB.GTLIN**

Call parameters		**Return parameters**	
D1		D1	???
D2		D2.L	???
D3		D3.W	Nbr of parameters fetched
D4		D4	???
D6		D6	???
A0		A0	???
A1		A1	Pointer to RI stack (rel. A6)
A2		A2	???
A3	Pointer to name table entry for 1st parameter (rel. A6)	A3	Preserved
A4		A4	Preserved
A5	Pointer 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 S*Basic 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.

Vector $110 Initialise S*Basic procedures and functions **SB.INIPR**

Call parameters		**Return parameters**	
D1		D1	Preserved
D2		D2	???
		D3+	Preserved
A0		A0	Preserved
A1	Pointer to Procedure / Function table	A1	???
		A2+	Preserved

Error returns:

IMEM No room for table

SB.INIPR is used to link in a list of Procedures and Functions to be added to the S*Basic name table. Once added, the functions can be called from S*Basic in the same way as the Procedures and Functions built into the ROM.

The structure of the Procedure / Function 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, which can be calculated with the following formula:

```
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 S*Basic put Parameter **SB.PUTP**

Call parameters		**Return parameters**	
D1		D1	???
D2		D2.L	???
D3		D3.L	???
A0		A0	???
A1	Pointer to value to be assigned (rel. A6)	A1	???
A2		A2	???
A3	Pointer 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 S*Basic name table. For details of the value to be assigned see Section 9.10. 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	Compare two strings	**UT.CSTR**

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 equivalent to a lower-case letter.

SPACE

!"#$%&'()*+,-/:;<=>?@[\]^_£{|}~©

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:

Type 0	Made directly on a character by character basis
Type 1	Made ignoring the case of the letters
Type 2	Made using the value of any embedded numbers
Type 3	Both ignoring the case of letters and using the value of embedded numbers

File and variable name comparisons use type 1.
Basic <, <=, =, >=, > and <> operators use type 2.
Basic == (equivalence) operator uses type 3.

Vector	$CA	Write error message to channel 0	**UT.WERSY**
	$CC	Write error message to given channel	**UT.WERMS**

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+	All preserved

Error returns:

All the usual I/O errors

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.

Vector $CE Write an integer to ASCII and sent it to the defined channel **UT.WINT**

Call parameters		**Return parameters**	
D1.W	Integer parameter	D1	???
D2		D2.L	???
D3		D3.L	???
A0	Channel ID or 0	A0	Preserved
A1		A1	???
A2		A2	Preserved
A3		A3	Preserved

Error returns:

All the usual I/O errors

This routine ought usually to be called from user mode. It can be called in Supervisor mode if A0=0.

Vector $D0 Send a message to a channel **UT.WTEXT**

Call parameters		**Return parameters**	
D1		D1	???
D2		D2.L	???
D3		D3.L	???
A0	Channel ID	A0	Preserved
A1	Base of message	A1	???
A2		A2	Preserved
A3		A3	Preserved

Error returns:

All the usual I/O
Condition codes set (sometimes - see below for QDOS v. 1.03 and earlier)

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 QDOS 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 recommended, but it does seem to work!

16.1. Vectored Routines - numerical order with page reference

mem.achp	$00c0	Allocate space in Common HeaP	9
mem.rchp	$00c2	Return space to Common HeaP	11
opw.wind	$00c4	Open WINDow using name	12
opw.con	$00c6	Open CONsole	12
opw.scr	$00c8	Open SCReen	12
ut.wersy	$00ca	Write an ERror to SYstem window	20
ut.werms	$00cc	Write an ERror MeSsage	20
ut.wint	$00ce	Write an INTeger	21
ut.wtext	$00d0	Write TEXT	21
mem.llst	$00d2	Link into LiST	10
mem.rlst	$00d4	Remove from LiST	10
cv.datil	$00d6	DATE and time (6 words) to Integer Long [SMS]	1
mem.alhp	$00d8	ALlocate in HeaP	9
mem.rehp	$00da	REturn to HeaP	11
ioq.setq	$00dc	SET up a Queue in standard form	4
ioq.test	$00de	TEST a queue for pending byte / space available	4
ioq.pbyt	$00e0	Put a BYTe into a queue	4
ioq.gbyt	$00e2	Get a BYTe out of a queue	4
ioq.seof	$00e4	Set EOF in queue	4
ut.cstr	$00e6	Compare STRings	19
iou.ssq	$00e8	Standard Serial Queue handling	7
iou.ssio	$00ea	Standard Serial IO	7
cv.ildat	$00ec	Integer (Long) to DAte and Time string	3
cv.ilday	$00ee	Integer (Long) to DAY string	3
cv.fpdec	$00f0	Floating Point to ascii DECimal	3
cv.iwdec	$00f2	integer (word) to ascii decimal	3
cv.ibbin	$00f4	integer (byte) to ascii binary	3
cv.iwbin	$00f6	integer (word) to ascii binary	3
cv.ilbin	$00f8	integer (long) to ascii binary	3
cv.ibhex	$00fa	integer (byte) to ascii hexadecimal	3
cv.iwhex	$00fc	integer (word) to ascii hexadecimal	3
cv.ilhex	$00fe	integer (long) to ascii hexadecimal	3
cv.decfp	$0100	decimal to floating point	2
cv.deciw	$0102	decimal to integer word	2
cv.binib	$0104	binary ascii to integer (byte)	2
cv.biniw	$0106	binary ascii to integer (word)	2
cv.binil	$0108	binary ascii to integer (long)	2
cv.hexib	$010a	hexadecimal ascii to integer (byte)	2

cv.hexiw	$010c	hexadecimal ascii to integer (word)	2
cv.hexil	$010e	hexadecimal ascii to integer (long)	2
sb.inipr	$0110	INITialise PRocedure table	17
sb.gtint	$0112	GeT INTeger	16
sb.gtfp	$0114	GeT Floating Point	16
sb.gtstr	$0116	GeT STRing	16
sb.gtlin	$0118	GeT Long Integer	16
qa.resri	$011a	QL Arithmetic Reserve Room on stack	15
qa.op	$011c	QL Arithmetic Operation	13
qa.mop	$011e	QL Arithmetic Multiple Operation	13
sb.putp	$0120	PUT Parameter	18
iou.dnam	$0122	decode Device NAMe	5
md.read	$0124	read a sector [QL]	8
md.write	$0126	write a sector [QL]	8
md.verif	$0128	verify a sector [QL]	8
md.rdhdr	$012a	read a sector header [QL]	8

17. Things [EXT][SMSQ/E]

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.

17.1. Thing structures

17.1.1. Thing linkage format

Items from **TH_THING** onwards (inclusive) must be filled in by the initialisation 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

17.1.2. 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.

17.1.3. List of Things Header

THH_NEXT	$08	long	offset of next Thing in list (0 for last)
THH_EXID	$0c	long	extra ID

17.1.4. Executable Thing Header

THH_HDRS	$08	long	offset to start of header
THH_HDRL	$0c	long	size of header
THH_DATA	$10	long	data space
THH_STRT	$14	long	offset of start of code or 0 to start at (copy of) header

17.1.5. 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

17.2. 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 programer 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 S*Basic procedure to re-link them if removed, but otherwise no special problems present themselves.

Things 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
(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.

17.3. Thing vectors

Whilst it was initially foreseen that the routines to use Things would be set up as TRAP#1 calls for SMSQ/E, this never happened. Versions 2.03 onwards of the HOTKEY System II, and SMSQ/E, thus 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.

The way to use Things thus is as follows: Find the THING Thing entry, get the Thing vector from there and call that with D0 used in the usual way to determine which vector should be used. You can use the code given as an example in section 17.4 below, but it is strongly suggested that you use the “*ut_thjmp*” utility routine in the SMSQ/E sources to do that (in the util_gut_subdirectory).

In this case, your code to use and then free an extension thing could be as follows (it is supposed that the extension you want to use is called “INFO”):

```
      move.l      #'INFO',d2        ; extension to use
      lea         thing_name,a0     ; point to name of thing
      moveq       #-1,d3            ; wait forever
      moveq       #-1,d1            ; I will use the thing
      moveq       #sms.uthg,d0      ; signal “use thing” vector
      jsr         gu_thjmp          ; on return A2= ptr thg header, a1 to thg
      move.l      d0,d1             ; use thing call ok?
      bne.s       err_out           ; no, return error!
      move.l      a1,a0             ; this is an extension thing
      (possibly setup parameters to which A1 will have to point)
      jsr         thh_code(a0)      ; call extension thing
      move.l      d0,d5             ; keep error
      lea         thing_name,a0
      moveq       #sms.fthg,d0      ; free thing
      moveq       #-1,d1
      jsr         gu_thjmp          ; get vector and call with d0
      tst.l       d5                ; did extension call go ok?
      (...)
thing_name
      dc.w        thn_end-*-2
      dc.b        ‘THING NAME’
thn_end
      (...)
```

D0=$29

SMS.FTHG [SMSQ][EXT]

Free a thing

Call parameters		Return parameters	
D1	User job id	D1	Preserved
D2	Parameter	D2	Returned result
D3	Parameter	D3+	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

D0=$26 **SMS.LTHG [SMSQ][EXT]**

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
- 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.

D0=$2B **SMS.NTHG** [SMSQ][EXT]

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
      JSR         gu_thjmp          ; jump via 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
```

D0=$2C

SMS.NTHU [SMSQ][EXT]

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       A0,A0                 ; start with first usage block
SLOOP
      MOVEQ       #SMS.NTHU,D0          ; find next user
      JSR         gu_thjmp              ; jump via vector:
      MOVE.L      D0,-(SP)
      BSR         proc                  ; process this user
      MOVE.L      (SP)+,D0              ; was there another Thing?
      BEQ.S       SLOOP                 ; yes!
SDONE
```

D0=$27 **SMS.RTHG** **[SMSQ][EXT]**

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 some Thing is required.

The Thing linkage block will have been returned to the common heap if this call succeeds.

D0=$28

SMS.UTHG [SMSQ][EXT]

Use Thing

Call parameters		Return parameters	
D1	Job ID	D1	Job ID
D2	Parameter or Extension ID	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
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.

D0=$2A

SMS.ZTHG [SMSQ][EXT]

Zap Thing

Call parameters		Return parameters	
D1		D1	Preserved
D2		D2	Preserved
D3		D3+	All preserved
A0	Name of thing to zap	A0	Preserved
A1		A1	???
A2		A2+	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.

17.4. Thing Entry Points

17.4.1. TH_ENTRY

Entry point is for calling from user mode. Under QDOS, all calls to SMS.ZTHG must be made in user mode, as must calls to FTHG on behalf of another Job.

17.4.2. 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.

17.4.3. 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
```

17.5. 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.

The 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.

17.5.1. 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

17.5.2. Level 1 Extension Thing Parameter Definition

The parameters for an 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.

17.5.3. Call Values and Keys

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 the latter has a numerical value. However, it appears that keys don't exist in the current Thing mechanisms.

thp.key	$0001	Key NOTE : doesn't exist.
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

thp..opt	12	Bit set if parameter optional
thp..nnl	11	Bit set if null parameter is negative (-1)

17.5.4. Pointer Parameter Usage

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.

thp..pointer	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.pointer	$8000	Call or return parameter (specified by calling code)

17.5.5. Optional Parameter

If the parameter is optional, then the optional bit should be set (or the word is ORed with the optional key value).

thp..opt	12	Bit set if parameter is optional
thp.opt	$1000	Optional

17.5.6. Array Parameter

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.

thp..arr	11	Bit set for array
thp.arr	$0800	Array

17.5.7. Parameter Types

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.

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

17.5.8. 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
```

17.5.9. 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..pointer** 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.

17.5.10. 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 S*Basic 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.

17.5.11. 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.

17.5.12. When to Use Extension Things

There are many ways of extending the 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.).

17.6. 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 variable	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 variable	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 variable	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 so on.

It must also return the Thing linkage block to the heap.

18. Keys

The following Section contain keys for various features of QDOS and SMSQ/E. These keys provide a definition for several of the data structures within QDOS and SMSQ/E.

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 S*Basic.

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 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 S*Basic' 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	POLIed 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		Default value
sys_rtim	$008E	Word	Repeat TIMe (20ms units)
sys.rtim	2		Default value

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_stiu	$00A6	Byte	msb Sector transfer in use [SMSQ]
sys_mtyp	$00A7	Byte	Machine TYPe / emulator type [SMS,ST]
sys.mtyp	$1E		Machine ID bits
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.mq40	$11		Q40/Q60
sys.mq68	$12		Q68
sys.mq0	$13		Qzero
sys.java	$14		SMSQmulator
sys.mtt	$18		TT
sys.mqem	$1A		Q-Emulator
sys.mqxl	$1C		QXL
sys.qpc	$1E		QPC
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 or QL mode LCD
sys.mvga	%11000000		VGA
sys.maur	%10000000		Aurora
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

Filing system defaults

sys.defo	$70		Offset to make defaults <$80
sys_prgd	$00AC	Long	Pointer to PRoGram Default [EXT][SMSQ]
sys_datd	$00B0	Long	Pointer to DATa Default [EXT][SMSQ]
sys_dstd	$00B4	Long	Pointer to DeSTination Default [EXT][SMSQ]
sys_thgl	$00B8	Long	Pointer to THinG List [EXT][SMSQ]
sys_psf	$00BC	Long	Primary stack frame pointer [SMSQ]
sys_200i	$00C0	Byte	200 Hz in service/interrupt 2 in service [SMSQ]

sys_50i	$00C1	Byte	50 Hz in service [SMSQ][Atari][Qx0]
sys_qlsd	$00C1	Byte	QLSD hardware in use [QL][EXTt]
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 suppressed timer [SMSQ]
sys.casup	26	Byte	25 full ticks
sys_iopr	$00CA	Word	I/O priority [SMSQ]
sys_cbas	$00CC	Long	Current basic (copy of sys_jbpt) [SMSQ]
sys_fpu	$00D0	16 Bytes	[SMSQ] (seem unused but do not touch)
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_klock	$00E4	Byte	Key lock [SMSQ]
sys..shk	0		Suppress HOTKEY
sys..ssf	1		Suppress screen freeze
sys..ssq	2		Suppress switch queue
sys..sbk	4		Suppress BREAK
sys..ssr	6		Suppress keyboard soft reset
sys..shr	7		Suppress keyboard hard reset
sys_mtick	$00E6	Word	Mini tick counter [SMSQ]
sys_klnk	$00E8	Long	Pointer to keyboard linkage [SMSQ/E]

Fixed filing system working area [QL, Q68]

sys_filw			$00EE to $0100
sys_cdiu	$00EE	Byte	Q68 flag for card in use [Q68]
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_q8ct	$00F0	Word	Q68 card type for cards 1 & 2 (a byte each) [Q68]
sys..q8un	$80		undetermined (card uninitialised)
sys..q8sd	9		simple SD card
sys..q8hc	0		SDHC card (or higher)
sys_mdst	$00F8	8*Byte	Status: 0=no pending ops [QL]

Filing system tables

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+, SMSQ, not SMS2]
sys_xtab	$0146	Long	Pointer to XLATE table [QDOS V1.10+, SMSQ, not SMS2]
sys_erms	$014A	Long	Pointer to (QDOS) error message table [QDOS V1.10+, SMSQ, not SMS2]
sys_mstab	$014E	Long	Pointer to (SMSQ) message table [SMSQ]. This is a pointer to a 256 long word table of pointers to message groups. All undefined message groups have a zero pointer.
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_rthg	$017D	Byte	use RECENT Thing (<>0 if yes) [SMSQ/E 3.24+]
sys_xdly	$017E	Byte	Suspend delay after executing another job [SMSQ/E 3.13+]

sys_ouch	$017F	Byte	Ouch flag (currently used to activate SGC debug) [SMSQ]
sys_top	$0180		Top of System Variables - 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

This table is for the variables of QDOS Superbasic. There are (some slight, some not so slight) differences with the variables for SMSQ/E.

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 number of window lines the 2 words immediately following this will be overwritten on changing **lenln** and **maxln**
bv_totln	$A6	Word	Number of window lines so far
bv_auto	$AA	Byte	Whether auto/edit on ($FF) or off (0)
bv_print	$AB	Byte	Print from **prtok** ($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	Position 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. SBasic Variables [SMSQ/E]

This table is for the variables of SMSQ/E SBasic. There are (some subtle, some not so subtle) differences with the variables for QDOS.

sb_bufft	-$64	long	input (etc) buffer top
sb_cmdlt	-$5c	long	command line top
sb_srcet	-$54	long	source program top
sb_nmtbt	-$4c	long	name table top
sb_nmlst	-$44	long	name list top
sb_datal	-$3c	long	pointer to list of data blocks
sb_chant	-$34	long	channel table top
sb_retst	-$2c	long	return stack top

sb_backl	-$20	long	parser backtrack stack limit
sb_grphl	-$18	long	parser graph stack limit
sb_chkhp	-$14	long	address of check heap patch
sb_arthl	-$10	long	arithmetic stack limit
sb_sbjob	-$0c	long	SBASIC Job
sb_prstl	-$08	long	processor stack limit
sb_flag	-$04	long	SBASIC flag
sb.flag	'SBAS'		
sb.toffp	-$68		top pointer offset from pointer
sb.loffp	-$68		limit pointer offset from pointer
sb.bofpu	-$04		base pointer offset from pointer (upwards)
sb.bofpd	$04		base pointer offset from pointer (dn stacks)
sb.dnspr	$10		spare space above downward stacks
sb_buffb	$00	long	input (etc) buffer base
sb_buffp	$04	long	... and pointer
sb_cmdlb	$08	long	command line (parsed) buffer base
sb_cmdlp	$0c	long	... and pointer
sb_srceb	$10	long	source program base
sb_srcep	$14	long	... and pointer
sb_nmtbb	$18	long	name table base
sb_nmtbp	$1c	long	... and pointer
sb_nmlsb	$20	long	name list base
sb_nmlsp	$24	long	... and pointer
sb_datab	$28	long	data area base (first block)
sb_datap	$2c	long	... sb_datap-sb_datab is total area allocated
sb_chanb	$30	long	channel table base
sb_chanp	$34	long	... and pointer
sb_retsb	$38	long	return stack base
sb_retsp	$3c	long	... and pointer
sb_backp	$48	long	parser backtrack stack pointer
sb_backb	$4c	long	... and base
sb_grphp	$50	long	parser graph stack pointer
sb_grphb	$54	long	... and base
sb_arthp	$58	long	arithmetic stack pointer
sb_arthb	$5c	long	... and base
sb_prstp	$60	long	processor stack pointer
sb_prstb	$64	long	... and base
sb_line	$68	word	line number
sb_stmt	$6c	byte	statement number
sb_cont	$6d	byte	set to continue, 00 to stop
sb_cmdst	$6e	byte	command line statement number
sb_cmdl	$6f	byte	set if command line
sb_cmdt2	$70	word	the offset from cmdst to token (or 0)
sb_frdat	$72	long	free space pointer for data area
sb_pmipt	$76	long	program interface pointer
sb_pmidt	$7a	long	program interface data
sb_pmint	$7e	byte	program interface flag
sb.pmint	$ff		
sb.pmist	$01		
sb_clc0	$7f	byte	set if channel zero to be closed on run
sb_rand	$80	long	random number
sb_cmdch	$84	long	command channel
sb_nline	$88	word	next line
sb_nstmt	$8a	byte	next statement
sb_actn	$8c	word	action number
sb.clear	$00		

sb.new	$02		
sb.stop	$04		
sb.run	$06		
sb.lrun	$08		
sb.load	$0a		
sb.mrun	$0c		
sb.merge	$0e		
sb.cont	$10		
sb.nact	$12		
sb_edt	$8e	byte	
sb.edt	$ff		program edited
sb.edtn	$80		new names have been set
sb.edtp	$01		program has been pre-compiled
sb_break	$8f	byte	msb clear if break
sb_cstmt	$91	byte	continue statement
sb_cline	$92	word	continue line
sb_dline	$94	word	current data line
sb_dstmt	$96	byte	current data statement
sb_ditem	$97	byte	current data item
sb_cmppg	$98	byte	compile program as well as command line
sb_auto	$aa	byte	set for auto line number
sb_pline	$ab	byte	set to print expanded line
sb_edlin	$ac	word	next edit line
sb_edinc	$ae	word	edit increment
sb_flags	$b0	long	flags
sb_cinst	$b0	byte	msb set if case dependent instr
sb_redo	$b8	byte	clear return stack and redo procedure / function
sb_lsfil	$ba	word	fill list window to here?
sb_rtmde	$bc	byte	ret stack mode (1 = line/statement no)
sb_colrm	$bd	byte	colour specification mode (0 or iow.papx)
sb_wherr	$be	word	when error clause line number
sb_wheiu	$c0	byte	when error in use
sb_inint	$c1	byte	in interpreter
sb_erno	$c2	long	error number
sb_eline	$c6	word	error line
sb_estmt	$c8	byte	error statement
sb_pcerp	$cc	long	error position during parsing / compiling
sb_pcern	$d0	long	error number during parsing compiling
sb_hichn	$d4	long	command line history channel ID
sb_qlibe	$dc	*****	QLiberator error table
sb_cheap	$e0	*****	common heap pointer
sb_qlibc	$e8	*****	QLiberator configuration
sb_qlibr	$ec	*****	QLiberator runtimes
sb_zero	$fc		zero
sb_dmbuf	$100		dummy buffer

Note: I deliberately left out the SBasic variables which have more to do with compiling the code .

18.5. Basic channel definitions and tokens

18.5.1. Offsets on BASIC Channel Definitions

The following Section gives the format of an entry in the S*Basic channel table. These entries can be monitored or modified by user-defined S*Basic 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

18.5.2. BASIC Token Values

The following Section defines the token values used for the internal storage of a S*Basic 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

In S*Basic, a literal number is represented as a 6 bytes floating point number by $Feee:mmmmmmmm with $0eee:mmmmmmmm (eee: exponent, mmmmmmmm: mantissa) being the actual floating point number (each e and m is one nibble).

[**SMSQ/E**] In addition, SBASIC allows for integer literal numbers to be entered as a binary or hexadecimal literal in the SBasic code if preceded by a % or a $, respectively. Their values are stored as floating point numbers in the tokenised program, the same as for literal decimal numbers, except that the token values are $Deee for binary, and $Eeee for hexadecimal.

18.6. 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. S*Basic 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
jcb_wflg	$0017	Byte	Set if there is a job waiting on completion of this one
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 I/O 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.

18.7. Slave 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

18.8. 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	I/O 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

18.9. File System Definition Blocks

18.9.1. 18.Standard channel block for filing system

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 Bytes spare [DD2]
chn_fend	$00A0		File system channel end [DD2]

chn_nchk	$00B0	Long	no 'in use' check in OS open code... [SMSQ/E]
chn.nchk	'NCHK'		... if set to this value

18.9.2. 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

18.9.3. 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 Byte	Microdrive sector map
md_lsect	$226	Word	Number of last sector allocated
md_pendg	$228	$100 Word	Map of pending operations - a word for each sector
md_end	$428		

18.9.4. Other Filing System Physical Definition Block [SMSQ][EXT]

iod_ftyp	$0023	byte	Format TYPe
iod.ql5a		0	0=QL5A
iod.qlwa		1	1=QLWA
iod.qlrd		2	2=QL ROM disk
iod.drct		-1	-ve direct
iod_rdln	$0024	long	Root Directory LeNgth
iod_rdid	$0028	word	Root Directory file ID
iod.rdid		1	implicit value for QL5A disks
iod_allc	$002a	word	ALLoCation size (bytes)
iod_totl	$002c	long	TOTaL allocation units
iod_free	$0030	long	FREE allcation units
iod_hdrl	$0034	long	inbuilt file HeaDeR Length
iod_rdfs	$0038	long	Root Directory First Slave block
iod_drst	$003c	byte	DRive Status : 0 unused, -ve OK
iodd.mod	1		medium modified (format or direct sector)
	other +ve		drive specific (error) flags
iod_wprt	$003d	byte	set if Write PRoTected
iod_sctl	$003e	word	SeCTor Length (for direct sector access)
iod_map	equ	$0040	long
iod_remv	$0044	long	pointer to def block / map remove code.
iod_xsbs	$0048	word	eXtra Slave Blocks required for each sector

iod.plen	$50	length of standard part of physical definition

18.10. Device Driver Linkage Block

for details refer to Section 7.1

iod_sqfb	-$08	Long	SMSQ I/O facility bit
iod..ssr	0		Bit set for serial
iod..swi	1		Bit set for window operations
iod..sfi	2		Bit set for filing system operations
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 I/O 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	I/O 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	Device Name (current USage)
iod_dnam	$42	String	Device NAMe [SMSQ]. These two are standard SMSQ/E strings of 1 to 4 characters. Beware : they may be used by some drivers for other information (eg. NET device).

18.10.1. Screen Driver Data Block Definition

sd_xmin	$18	Word	Window top LHS
sd_ymin	$1A	Word	
sd_xsiz	$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

18.10.2. 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		

18.10.3. 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		

18.11. 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 (MS bit)
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	

18.12. 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. (NB TOS = Top Of Stack, NOS = Next On Stack)

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	

18.13. 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]

18.14. 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	Microdrive 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 read 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 is 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

18.15. 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. Traps are sorted by the D0 key.

18.15.1. 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 S*Basic area
sms.rmpa	$17	Release space in S*Basic area
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 translation and error messages
sms.xtop	$25	eXTernal Operation [SMSQ]

sms.iopr	$2E	IO PRiority [SMSQ]
sms.cach	$2F	CACHe handling [SMSQ]
sms.lldm	$30	Link in Language Dependent Module [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]
sms.wevt	$3B	Wait for event [SMSQ]

18.15.2. 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.krn	$05	Key for ReNaMe [DD2]

18.15.3. Trap 3 Keys (I/O Traps)

do.io	3	Trap #3
do.relio	4	Trap #4

lob.test	$00	TEST input
lob.fbyt	$01	Fetch BYTe from input
lob.flin	$02	Fetch LINe from input
lob.fmul	$03	Fetch MULtiple characters/bytes
lob.elin	$04	Edit LINe of characters
lob.sbyt	$05	Send BYTe to output
lob.suml	$06	Send a string of untranslated bytes [SMSQ/E]
lob.smul	$07	Send MULtiple bytes
low.xtop	$09	eXTernal OPeration on screen
low.pixq	$0A	PIXel coordinate Query
low.chrq	$0B	CHaRacter coordinate Query
low.defb	$0C	DEFine Border
low.defw	$0D	DEFine Window
low.ecur	$0E	Enable CURsor
low.dcur	$0F	Disable CURsor

low.scur	$10	Set CURsor position (character coordinates)
low.scol	$11	Set cursor COLumn
low.newl	$12	put cursor on a NEW Line
low.pcol	$13	move cursor to Previous COLumn
low.ncol	$14	move cursor to Next COLumn
low.prow	$15	move cursor to Prevous ROW
low.nrow	$16	move cursor to Next ROW
low.spix	$17	Set cursor to PIXel position
low.scra	$18	SCRoll All of window
low.scrt	$19	SCRoll Top of window (above cursor)
low.scrb	$1A	SCRoll Bottom of window (below cursor)
low.pana	$1B	PAN All of window
low.panl	$1E	PAN cursor Line
low.panr	$1F	PAN Right hand end of cursor line
low.clra	$20	CLeaR All of window
low.clrt	$21	CLeaR Top of window (above cursor)
low.clrb	$22	CLeaR Bottom of window (below cursor)
low.clrl	$23	CLeaR cursor Line
low.clrr	$24	CLeaR Right hand side of cursor line
low.font	$25	set / read FOuNT (font U.S.A.)
low.rclr	$26	ReCoLouR a window
low.spap	$27	Set PAPer colour
low.sstr	$28	Set STRip colour
low.sink	$29	Set INK colour
low.sfla	$2A	Set FLash Attribute
low.sula	$2B	Set UnderLine Attribute
low.sova	$2C	Set OVerwrite Attributes
low.ssiz	$2D	Set character SIZe
low.blok	$2E	fill a BLOcK with colour
low.donl	$2F	DO a pending newline
log.dot	$30	draw (list of) DOTs
log.line	$31	draw (list of) LINEs
log.arc	$32	draw (list of) ARCs
log.elip	$33	draw ElIIPse
log.scal	$34	set graphics SCALe
log.fill	$35	set area FILL
log.sgcr	$36	Set Graphics CuRsor position
lof.chek	$40	CHEcK all pending operations on file
lof.flsh	$41	FLuSH all buffers
lof.posa	$42	set file POSition to Absolute address
lof.posr	$43	move file POSition Relative to current position
lof.minf	$45	get Medium INFormation
lof.shdr	$46	Set file HeaDeR
lof.rhdr	$47	Read file HeaDeR
lof.load	$48	(scatter) LOAD file
lof.save	$49	(scatter) SAVE file
lof.rnam	$4A	ReNAMe file [EXT, DD2]
lof.trnc	$4B	TRuNCate file to current position [EXT, DD2]
lof.date	$4C	set or get file DATEs [EXT,DD2]
lof.mkdr	$4D	MaKe DiRectory [DD2]
lof.vers	$4E	set or get VERSion [DD2]
lof.xinf	$4F	get eXtended INFormation [DD2]
low.papp	$50	Set paper colour (palette) [SMSQ/E]
low.strp	$51	Set strip colour (palette) [SMSQ/E]
low.inkp	$52	Set ink colour (palette) [SMSQ/E]
low.borp	$53	Set border colour (palette) [SMSQ/E]
low.papt	$54	Set paper colour (24 bit) [SMSQ/E]
low.strt	$55	Set strip colour (24 bit) [SMSQ/E]
low.inkt	$56	Set ink colour (24 bit) [SMSQ/E]
low.bort	$57	Set border colour (24 bit) [SMSQ/E]
low.papn	$58	Set paper colour (native) [SMSQ/E]

IOW.STRN	$59	Set strip colour (native) [SMSQ/E]
IOW.INKN	$5A	Set ink colour (native) [SMSQ/E]
IOW.BORN	$5B	Set border colour (native) [SMSQ/E]
IOW.BLKP	$5C	Fill block with colour (palette) [SMSQ/E]
IOW.BLKT	$5D	Fill block with colour (24 bit) [SMSQ/E]
IOW.BLKN	$5E	Fill block with colour (native) [SMSQ/E]
IOW.PALQ	$60	Define QL colour palette [SMSQ/E]
IOW.PALT	$61	Define 8-bit colour palette [SMSQ/E]
IOW.SALP	$62	Set the alpha blending weight for window

Please note ; there also exist keys higher than $62. They are for pointer-driven CON devices. Please refer to the QPTR manual.

Timeout keys

no.wait	0
forever	-1

18.16. 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

18.17. 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	$0	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 tried
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..pointer	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..pointer
thp..arr	11	Bit set for array - thp..pointer
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..pointer**+1<<**thp..cal**+1<<**thp..ret**	Updated parameter
thp.call	1<<**thp..pointe**r+1<<**thp..cal**	Call parameter
thp.ret	1<<**thp..pointer**+1<<**thp..ret**	Return parameter
thp.pointer	1<<**thp..pointer**	Call or return parameter

18.18. 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 previous)
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 keyboard with previous string from buffer
hki.stbf	-4		Stuff keyboard queue from buffer
hki.stuf	-2		Stuff keyboard queue with string
hki.cmd	0		Pick S*Basic 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_pointer	$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

18.19. Keys for format of pointer device driver definition block

(note : B = byte ; W = word ; L = longword)

pt_lext	$00		the usual links and I/O routines for a device driver)
pt_aext	$04	L	
pt_lpoll	$08	L	
pt_apoll	$0c	L	
pt_lschd	$10	L	
pt_aschd	$14	L	
pt_liod	$18	L	
pt_aio	$1c	L	
pt_aopen	$20	L	
pt_aclos	$24	L	
pt_wman	$28	L	pointer to window manager version / vector
pt_wmove	$2c	B	top window move / resize status (0, $80, $81)
pt_keyrw	$2d	B	last keyrow
pt_xicnt	$2e	W	count of external interrupts
pt_accel	$30	W	accelerator (keyboard cursor)
pt_kaccl	$32	W	accelerator constant (d=1+ticks/kaccl)
pt.kaccl	8		
pt_xaccl	$34	W	x mouse accelerator (x=x+(xinc*(xaccl+xinct)/(xaccl+1)
pt.xaccl	3		
pt_yaccl	$36	W	Y mouse accelerator
pt.yaccl	3		

pt_npos	$38	L	new position (set by poll)
pt_nxpos	$38	W	new x position
pt_nypos	$3a	W	new y position
pt_pos	$3c	L	current pointer position
pt_xpos	$3c	W	current x position
pt_ypos	$3e	W	current y position
pt_inc	$40	L	positional increment
pt_xinc	$40	W	x
pt_yinc	$42	W	y
pt_addr	$44	L	current (or old) screen address of pointer
pt_bstat	$48	B	button status
ptb.bnce	-2		... button bouncing
ptb.up	0		... button up
ptb.pres	1		... button pressed
pt_bsupp	$49	B	button suppressed
ptb.psup	-1		... press suppressed
ptb.ssup	1		... stroke suppressed
pt_bpres	$4a	B	button pressed (space/HIT or ENTER/DO)
pt_bcurr	$4b	B	button currently pressed (becomes stroke)
pt_wake	$4c	B	external interrupts to wake up pointer
pt.wake	3		
pt_relax	$4d	B	relaxation time before pointer falls onto kbd input
pt.relax	25		
pt_reltm	$4e	B	relaxation timer
pt_kprtm	$4f	B	keypress timer
pt_pstat	$50	B	pointer status - 0 visible, -ve pending, +ve immovable
pt.supky	1		keyboard input - suppress forever
pt.show	-1		show pointer next time around
pt.supio	-128+20		IO - if no ptr read within 20 ticks - suppress
pt.supmd	1		suppress for mode
pt.supcc	1		suppress on ctrl c
pt.supsr	pt.supio		save or restore - suppress as IO
pt.clrdq	20		wait before clear dummy queue
pt_pmode	$51	B	previous mode
pt_schfg	$52	B	scheduler flag, true if scheduler updated pointer record
pt_change	$58	L	address of screen size / colour depth change routine
pt_pfrx	$5c	W	x position, fractional part
pt_pfry	$5e	W	y position, fractional part
pt_psprt	$60	L	pointer to sprite
pt_spsav	$64	L	sprite save area size in pixels
pt.spspx	64		64x64 pixels
pt.spspy	64		
pt_head	$68	L	head pointer to linked list of primary windows
pt_tail	$6c	L	tail pointer to linked list of primary windows
pt_dumq1	$80	L	pointer to first dummy queue
pt_kqoff	$84	L	offset of keyboard queue from (a0)
pt_copen	$88	L	pointer to standard console open
pt_lstuf	$8c	B	last button press for stuffer
pt_stuf1	$8d	B	first character to stuff
pt.stuf1	$ff		
pt_stuf2	$8e	B	second character to stuff
pt.stuf2	'.'		
pt_offscr	$8f	B	bits set if pointer on limits of screen
pt..otop	3		
pt..obot	2		
pt..oleft	1		
pt..oright	0		
pt_rec	$90	L	current pointer record

pt_cchad	$a8	L	current channel address (the window the pointer is in)
pt_cwstt	$ac	B	current window status +1 wrong mode, -1 key input
pt_cwbsy	$ad	B	current window busy count
pt.cwbsy	10		number of scheduler loops before true busy
pt_supcr	$ae	B	suppress cursor key stuffing
pt_bpoll	$af	B	buttons set on poll
pt_kypol	$af	B	same for ptr_gen- buttons set on poll
pt_svers	$b0	B	dynamic Sprite VERSion
pt_mvers	$b1	B	Max count for this sprite VERSion
pt_randi	$b2	W	RANDom number Initial value (for spray)
pt.randm	$e72b		RANDom amount to Multiply by
pt.randa	3		RANDom amount to Add on
pt_spbuf	$b4	L	pixel SPray BUFfer
pt_spbsz	$b8	L	pixel SPray Buffer SiZe
pt_hbase	$bc	L	hardware bas
pt_kyrwr	$c0	L	pointer to keyrow read code
pt_ptrok	$c4	L	points to OK
pt_nuldr		pt_ptrok-pt_aclos+pt_liod	
pt_aiorm	$c8	L	pointer to ROM I/O routine
pt_aoprm	$cc	L	pointer to ROM openroutine
pt_aclrm	$d0	L	pointer to ROM close routine
pt_romdr		pt_aiorm-pt_aio+pt_liod	
pt_ochad	$d4	L	old current channel address
pt_dmode	$dc	B	display mode
pt_ptlim	$dd	B	set if pointer movement limited (temporary)
pt_ckeyw	$de	B	clear if cursor keys move pointer on a window basis
pt_ckey	$df	B	clear if cursor keys move pointer
pt_minxy	$e0	W	x pointer limit
pt_maxxy	$e4	W	y pointer limits

(old names)

pt_sbase	*$e8*	*screen base*
pt_bytes	*$ec*	*bytes per screen*
pt_bytel	*$f0*	*bytes per line*
pt_ssize	*$f2*	*screen size*
pt_ssizx	*$f2*	
pt_ssizy	*$f4*	

(new names)

pt_scren	$e8	L	address of visible SCREeN
pt_scrsz	$ec	L	SCReen SiZe in bytes
pt_scinc	$f0	W	SCreen row INCrement
pt_xscrs	$f2	W	X SCReen Size in pixels
pt_yscrs	$f4	W	Y SCReen Size in pixels
pt_xtotl	$f6		X total pixels
pt_xvref	$f8	W	X visible reference for xtotl
pt_ytotl	$fa	W	Y total pixels
pt_yvref	$fc	W	Y visible reference for ytotl
pt_frate	$fe	W	frame rate
pt_bgstp	$102	W	background stipple
pt_bgclm	$104	L	background colour mask
pt_sfnt1	$108	L	pointer to Standard FoNT 1
pt_sfnt2	$10c	L	pointer to Standard FoNT 2
pt_spcch	$110	L	pointer to sprite cache
pt_palql	$114	L	pointer to QL palette
pt_pal256	$118	L	pointer to 256 colour palette

pt_pspck	$11c	L	pointer to pointer sprite for checking
pt_bgcld	$120	L	background colour definition
pt_wdata	$124	L	pointer to WMAN data area
pt_ident	$128	L	identifier
pt.ident	'PTR2'		
pt_drtab	$12c	5*L	driver installation routine addresses (5*long)
pt_cdpth	$140	B	current colour depth
pt_cdtab	$141	5*B	modes for each resolution (5*byte)
pt_sstb	$146	L	pointer to system sprite table
pt_asprt	$14a	Float	float pixel aspect ratio (float = 6 bytes)
pt_vecs	$150	L	pointer to block with vectored routines
pt_cjob	$154	L	pointer to cursor sprite job table
pt_swwin	$158	W	current window no in pile during CTRL+C switch
pt_spszy	$15a	W	pointer sprite save area y size
pt_bgdat	$15c	L	pointer to memory for background I/O data
pt_end	$160		length of pointer linkage block

18.20. Hard disk format: QLWA

QL formatted hard disks, as well as the “QXL.WIN” container files which mimic such disks, use a file allocation table (FAT) and cluster based approach. Sector 0 of the disk contains general information about the disk itself and the start of the FAT. Note that in QL parlance, the FAT is a “map”, and a cluster is a “group”.

Mnemonic	Offset	Size	Description
qwa.map	$0		first map sector
qwa_id	$0000	long	ID
qwa.id	'QLWA'		standard ID for this kind of disk
qwa.pflg	$01515741		partition flag
qwa_name	$0004	string	up to 20 characters space padded name. This is a standard string, i.e. preceded by a length word
qwa_spr0	$001a	2 bytes	spare - set to zero
qwa_uchk	$001c	long	update check (removable media only)
qwa_intl	$0020	word	interleave factor (0 SCSI)
qwa_sctg	$0022	word	sectors per group
qwa_sctt	$0024	word	sectors per track (0 SCSI)
qwa_trkc	$0026	word	tracks per cylinder (number of heads) (0 SCSI)
qwa_cyld	$0028	word	cylinders per drive
qwa_ngrp	$002a	word	number of groups
qwa_fgrp	$002c	word	number of free groups
qwa_sctm	$002e	word	sectors per map
qwa_nmap	$0030	word	number of maps
qwa_free	$0032	word	first free group
qwa_root	$0034	word	root directory number
qwa_rlen	$0036	long	root directory length
qwa_fcyl	$003a	word	first cylinder number (ST506)
qwa_fsct	$003a	long	first sector for this partition (SCSI)
qwa_park	$003e	word	park cylinder
qwa_gmap	$0040		group map: each entry is number of next group, or zero if there is no next group

19. SMSQ/E

This chapter deals specifically with SMSQ/E. It is a separate chapter so that you can see the advantages of SMSQ/E at one glance. All the descriptions listed here will be referenced from the other chapters and additional traps are also put into the right chapters 13 to 15.

As SMSQ/E is a growing system which will be expanded depending on user's requirements, this manual can reflect the features of SMSQ/E at the current situation only. It is quite possible that a number of features are not available on earlier versions of SMSQ/E. At the time of writing, the version of SMSQ/E is V3.29. 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 the 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.

19.1.2.1. 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.

19.1.2.2. 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: no shift and no ctrl ; shift and ctrl; ctrl and no shift; shift and ctrl.

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.

19.1.2.3. 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 positive20
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)
39	I	Italian (in Italy)
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.

19.1.6. System Variables

See section 18.2 above, variables from $0144 to $014e and $0168 to $16C.

19.1.7. Additional Trap #1 Calls

There are a number of SMSQ OS Trap #1 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 (pointed to by A1) into the language dependent module list.

Trap #1 D0=$31

SMS.LENQ

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.

Trap #1 D0=$32

SMS.LSET

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.

Trap #1 D0=$33

SMS.PSET

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.

Trap #1 D0=$34 **SMS.MPTR**

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.

19.2. Additional Trap #3 calls

SMSQ/E introduced additional trap #3 calls. These are documented in the general description of all Trap#3 calls. As a reminder, the traps introduced are:

IOW.FONT	D0=$25	Set or reset the default system fount
IOB.SUML	D0=$06	Send a string of untranslated bytes

19.3. SMSQ Cache Handling

19.3.1. Principles

SMSQ is implemented on many 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.

19.3.1.1. 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 I/O 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 I/O 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.

19.3.1.2. 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 I/O 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 I/O 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.

19.3.1.3. 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.

19.3.1.4. 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.

19.3.2. 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.

19.3.3. 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,CA
```

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
$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
```

19.3.4. 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.

19.3.4.1. 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.

19.3.4.2. 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.

19.3.4.3. 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.

19.3.4.4. 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.

19.3.4.5. 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.

19.3.4.6. 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.

19.3.4.7. 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.

19.3.4.8. System Variables

sys_castat	$C8	word	MSB set if cache fully enabled
sys_casup	$C9	byte	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.

20. 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
      jsr         gu_thjmp          ; 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.se	$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!

Find a HOTKEY item **HK.FITEM**

Call parameters		**Return parameters**	
D1		D1.W	HOTKEY
D2		D2.W	HOTKEY number (-ve if off)
		D3+	Preserved
A1	HOTKEY item name	A1	Pointer 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.

Create the HOTKEY job **HK.CRJOB**

Call parameters		**Return parameters**	
		D1+	Preserved
A3	Linkage block	A3	Preserved

Error returns:

All system errors related to jobs

Kill the HOTKEY job **HK.KJOB**

Call parameters		**Return parameters**	
		D1+	Preserved
A3	Linkage block	A3	Preserved

Error returns:

Always succeeds

Set or reset a HOTKEY

HK.SET

Call parameters			Return parameters	
D0.B	-1	Off		
	0	On		
	1	Reset		
	2	Set		
D1.W	New Key	Reset : D0=1	D1.W	HOTKEY
		Set : D0=2		
			D2+	Preserved
A1	Pointer to item (Set)			
	Pointer to key or name (Off, On, Reset)			
A2				
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.

Remove HOTKEY item

HK.REMOV

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

"DO" a HOTKEY item **HK.DO**

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

Set a string in the stuffer buffer **HK.STBUF**

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.

Get stuffer buffer contents

HK.GTBUF

Call parameters		**Return parameters**	
D0.B	Key: -1 = Previous string 0 = Current		
D2.W		D2.W	Length of string
		D3.W	Preserved
A1		A1	Pointer to characters
A2		A2	Preserved
A3	Linkage block	A3	Preserved

Error returns:

Always succeeds

Open and clear guardian window

HK.GUARD

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.

20.1.1. 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 keyboard buffer with previous string from buffer
hki.stbf	-4		Stuff keyboard queue from buffer
hki.stuf	-2		Stuff keyboard queue with string
hki.cmd	0		Pick S*Basic 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_pointer	$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).

20.1.2. Hotkey Vectors

The Hotkey vectors are in the Hotkey Thing. These are available in all HOTKEY System II versions.

21. The Button Frame [EXT]

The concept of the Button Frame (built into QPAC II) is described here. QPAC II is now freely available. Whilst the button Frame is not really part of SMSQ/E or QDOS, it was thought helpful to set out the documentation for it here.

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'
```

Use the Button Frame			**BT.USE**
Call parameters		**Return parameters**	
D1	User Job ID	D1	Job ID
D2.L	Button Size	D2.W	Button Origin
D3.L	0 New allocate		
	-ve For re-allocate		
		D3	Version
A0	Pointer 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.

Free the Button Frame **BT.FREE**

Call parameters		**Return parameters**	
D1	User Job ID	D1+	Preserved
D2.L	Pointer to name of Thing		
D3.L	New allocate		
A0	Pointer to Thing Name	A0	Preserved
A1		A1	Preserved
A2	Base of usage block or 0 for 1st one	A2	???
		A3+	Preserved

Error returns:

Always successful

22. The HOME Thing [EXT] [SMSQ/E]

The latest version of SMSQ/E has an inbuilt support for a "HOME directory Thing". This also exists, to a less integrated extent, for QDOS.

22.1. Purpose and facilities

22.1.1. Home directory

The HOME Thing implements "home directories". A home directory is the directory from which an executable file was executed. Thus, if you have a file called "fred_exe" in a directory "win1_progs_exec_" the home directory for that file will be "win1_progs_exec_".

22.1.2. Home Filename

The HOME Thing also provides for the "home filename" which is the combination of the filename and the home directory, thus making up the complete SMSQ/E filename - in the example above this would be "win1_progs_exec_fred_exe".

Both home directory and home filename are set up once and for all when the program starts, and are deleted when the program is removed. With one exception, they are immutable: once set, they may not be changed. They are just removed upon removal of the program itself.

22.1.3. Current Directory

The HOME Thing also implements a "current directory". This is inherited from the job that is setting up the home directory (in most cases the parent job). If the calling job does not have a current directory, a copy of the home directory is used instead.

The current directory can only point to a valid directory. Within that limit it may be set/reset or otherwise manipulated by the job itself.

22.1.4. Default Directory for named jobs

Finally, there is also a default home directory for jobs that are executed through other means, perhaps through file managers that don't use the HOME Thing, or, especially, through Hotkeys. Due to the enormous variations that can exist in situations where jobs are executed from Hotkeys, and while there is no problem when jobs are loaded from a file through a Hotkey, sometimes the job code is already in memory, but no job with the name exists until the Hotkey is actually pressed ("executable things"), sometimes the job executes immediately etc. In those circumstances, it will not always be possible to associate a job with a filename and directory.

It is, however, possible to set up a default home directory for jobs with a given name. When a job, for which a default directory was set up, executes, for example from a Hotkey, and tries to get at its home directory, a home directory will be set up for it automatically.

Thus, whenever a job tries to find its home/current directory/file and they can't be found directly because they haven't been defined for that job, a check is also made in the default list. If the job's name is in the default list, then an entry for that job, with that default filename, is made in the home directory list.

22.2. The HOME Thing under SMSQ/E and QDOS

For the home directory scheme to work, the cooperation of the operating system or file manager(s) is needed: Indeed, whenever a job is executed, whoever is doing this executing must explicitly set up the home directory for the job that is being executed. Here, there is a difference between SMSQE and QDOS.

22.2.1. SMSQ/E

As of version 3.11, SMSQ/E has the HOME Thing built in, and also support for it (see below for QDOS). Typically, on an SMSQ/E system, jobs will be started up through the EX(ec) command variants, through file managers such as QPAC II or through FileInfo. SBasic programs may also be loaded.

22.2.1.1. The EX(ec) etc commands

Whenever you use the EX command or any of its variants, the home directory for the job to be EXecuted will be set up automatically.

22.2.1.2. QPAC II and other file managers

QPAC II has already be altered to take the new HOME Thing into account. All other file managers will need to be changed to support the home directory. If you are a programmer and have programmed a file manager, further information is given below, showing you the code that needs to be implemented for this. If a file manager is in compiled basic (e.g. DiskMate) no further action will be necessary under SMSQ/E since the EX commands in SMSQ/E will do whatever is necessary.

22.2.1.3. FileInfo

FileInfo II has been modified to use the HOME Thing, and so has .the original FileInfo.

22.2.1.4. Basic

Under SMSQE, whenever you (q)load/(q)merge a basic program, the home directory for that basic program is set to the file just loaded. Thus Basic is again an exception - it is the only job for which the Home directory may change.

22.2.2. QDOS

There was a stand-alone version of the HOME Thing for QDOS users, which could be downloaded at:

http://www.lenerz.com/QLStuff

ATM this version is no longer available.

(Thus, QDOS systems can also profit the home directories set up from QPAC II and FileInfo, but support for the HOME Thing through the EX and LOAD commands will be non-existent, since that requires a change in these commands. The same is true for file managers that are compiled basic.

To load the HOME Thing under QDOS, use:

```
a=RESPR (file_length)
LBYTES <device>_home_bin,a
CALL a
```

or the LRESPR variants if your system has them.)

22.3. Using the HOME Thing

22.3.1. From SBasic

There are several new SBasic keywords for this.

22.3.1.1. Get the home directory

result$ = **HOME_DIR$(job_id)**

This function returns the home directory for the job given as job_id. To avoid programs stopping with an error if for some unimaginable reason the home directory cannot be found, this function returns an empty string if that error happens.

The job ID is optional. If it is omitted is defaults to -1, meaning the current job.

Example:

```
   ....
100 define procedure init
110 mydir$ = HOME_DIR$
   ....
```

22.3.1.2. Get the home filename

result$ = **HOME_FILE$(job_id)**

Same as for the home directory, but for the home filename.

22.3.1.3. Get the current directory

result$ = **HOME_CURR$(job_id)**

Same as for the home directory, but for the current directory.

22.3.1.4. Default names

HOME_DEF job_name$, file_name$

This sets a default filename for a job with the name given as first parameter. This is useful for "executable things", where no filename is readily available, or for file managers that haven't integrated calls to the HOME Thing. Please refer to the Section 22.2.1.4 above for more information on this.

With this keyword, you set up the default job name and filename that is to be used for the home/current file/directory.

Please note that the file_name$ parameter must indeed be a FILENAME, not a directory name.

Example:

```
HOME_DEF "SBasic", "dev1_SBasic_test_bas"
```

22.3.1.5. Get the version of the HOME Thing

result$ = **HOME_VER$**

22.3.2. From machine code

The HOME Thing implements various extensions - it is an extension thing.

There is some ready-to-use wrapper code in the SMSQ/E source tree, namely in the file "util_gut_home_asm", or you can make use of the following:

To use the extensions, in short, first you USE the thing (A0 = thing name, D2 = extension). You might want to make sure you use a call that returns the pointer to the thing linkage base in A2 and a pointer to the thing in A1 as these will be expected when calling the call routine. (If you have access to the SMSQE sources, a good vector for this is **gu_thjmp**). The name of the thing is, imaginatively, "HOME", and the names of the individual extensions are as given below.

After you used the thing, you then call the thh_code routine of the thing with A2 pointing to the linkage and A1 to a parameter list.

Each extension thus has its own parameter list. They are explained below for each extension:

GETH Get the home directory

GETF Get the home filename

GETC Get the current directory.

Call parameters		**Return parameters**	
D0		D0	0 or error
D2		D2	Buffer size needed (if **err.orng**)
A1	Pointer to parameter list	A1	Preserved
A2	Pointer to Thing linkage preserved	A2	Preserved

Error returns:

ITNF the job with the given ID doesn't have a home directory

ORNG the buffer is too small (see below)

The parameter list is as follows:

0(A1)	Long word	Job id of job for which info is to be gotten
4(A1)	Word	$A100 (corresponding to **thp.str+thp.ret)**
6(A1)	Word	Length of buffer for return string
8(A1)	Long word	Pointer to buffer for return string

The routine will return 0 if no error occurred, **err.itnf** if the job with the given ID doesn't have a home directory and **err.orng** if the buffer is too small for the entire directory / filename. In this latter case, the routine will not touch the given buffer but just return the needed size in D2.

SETD	Set the default directory for a given name.		
Call parameters		**Return parameters**	
D0		D0	0 or error
A1		A1	Preserved
A2	Pointer to parameter list	A2	Preserved

The parameter list is as follows:

0(A1)	Long word	$C1000000 (corresponding to **thp.str+thp**.call)
4(A1)	Long word	Pointer to string for jobname
8(A1)	Long word	$C1000000 (corresponding to **thp.str+thp**.call)
12(A1)	Long word	Pointer to string for filename

22.4. Setting up a home directory

Normally, jobs should not try to set up a home directory for themselves. This should be left to the system / file manager. When a job is started with the SMSQ/E EX, EW or any of the similar commands, this is done automatically. However, file manager writers may be interested in this information.

22.4.1. From S*Basic

HOME_SET job_id, device_and_file_name$

Set the home directory, home filename and current directory. You pass the thing the job ID of the job for which this is to be set up and the entire filename, including the device and directory. The thing extracts the home directory from the filename. If you want to set up the home directory for the current job, you may pass -1 as parameter.

Since there can only be one home directory for a job and since that can only be defined once, the keyword will give an error if the home directory is already set for this job. Otherwise, this keyword will set the home directory, the home file and the current directory.

This keyword exists mainly for testing purposes.

22.4.2. From Machine Code

SETH This sets the home directory

Call parameters		**Return parameters**	
D0		D0	0 or error
A1	Pointer to parameter list	A1	Preserved
A2	Pointer to Thing linkage preserved	A2	Preserved

Error returns:	
FDIU	job home directory is already in use
IJOB	the job doesn't exist
IPAR	no filename passed (empty string!)
any error from memory allocation routine	
any error from open file	

The parameter list is as follows:

0(A1)	Long word	Job ID of the job for which this is set
4(A1)	Long word	$C1000000 (corresponding to **thp.str+thp**.call)
8(A1)	Long word	Pointer to string for entire filename

The following is an example of a routine that can use this to SET the home name/dir (this uses the **gu_thjmp** routine from the SMSQE sources, which returns the correct values in A1 and A2). It is presumed here that, on entry D1 contains the job ID of the job for which the directory/file is to be set and that the filename for this job can be found at a label 'filename'.

```
homereg           reg   a0-a4/d0-d3
d1stak            equ   4           ; where D1 is on stack

sethome
      movem.l     homereg,-(sp)     ; keep my regs
      lea         home_name,a0      ; point to name of thing
      moveq       #-1,d3            ; wait forever
      moveq       #-1,d1            ; I will use the thing
      move.l      #'SETH',d2        ; extension in thing to use
      moveq       #sms.uthg,d0      ; use thing
      jsr         gu_thjmp          ; on return A2= ptr thg header, a1 to thg
      tst.l       d0                ; ok?
      bne.s       no_thg            ; no, ignore
      move.l      a1,a0             ; pointer to thing (!!!)
      move.l      d1stak(sp),d1     ; get job ID back
      sub.l       #12,sp            ; get some space
      move.l      sp,a1             ; and point to it
      move.l      d1,(a1)           ; insert ID of job
      move.l      #$c1000000,4(a1)  ; thp.call+thp.str
      lea         filename,a4       ; point to file/dirname to set
```

```
      move.l      a4,8(a1)          ; set pointer to this string
      jsr         thh_code(a0)      ; call extn thing
      add.l       #12,sp            ; reset stack
      lea         home_name,a0      ; now free thing, ignore error on call
      moveq       #sms.fthg,d0
      moveq       #-1,d1
      jsr         gu_thjmp          ; free thing
no_thg
      movem.l     (sp)+,homereg     ; ignore error
      (...)

home_name
      dc.w  4,'HOME'
```

23. The RECENT Thing [SMSQ/E]

The RECENT Thing maintains lists with the names of recently opened files so that you can find out what program recently opened files, and so that application programs may propose a list of the files the user recently opened :

- There is one general list, which contains the name of files recently opened, irrespective of the job which opened them.
- Then there is a list for each job that opened one or several files and which only contains the files opened by this job.

The RECENT Thing is an extension thing. To use it, configure a copy of your SMSQ/E file and reboot with that copy of SMSQ/E. You will see that you have a new thing, the RECENT Thing. You can use the compiled Basic program called dev8_extras_exe_show_recents (in the SMSQ/E sources) to have a look at recently opened files. A help file (dev8_extras_help_show_recents_txt) for this compiled basic program exists in the SMSQ/E sources. You might need some extensions if you want to recompile the basic program, they should normally be found at www.wlenerz.com/qlstuff.

Various ways are provided to obtain the list(s) from the thing. There is an assembler interface to the thing and the extensions, and various SBasic keywords that map onto the extensions. Both are explained below.

23.1. Concepts

The RECENT Thing is called directly from the system's open file trap, without any user intervention. Whenever a file is opened, its name is added to the general list and to the job's list, and then becomes the first element of these lists.

There is no provision to delete files from the lists. However, since every list only has a finite size, when it is full and a new file is added to it, this will push the most ancient file off the list and the newest on it. There is also the possibility to remove an entire list for a job.

The system's open file call tries to filter out calls to open a directory, so that a directory open call does not cause the name of the directory to be added to the list. The same is true for the SAVE file. You can configure the size of the lists (i.e. how many files it should contain).

23.1.1. The lists

Each list is implemented as a lifo buffer : the last (most recently) opened file will be the first in the list. There is one exception to this rule: The thing makes sure that a file only exists once in a list. So it checks the general list and the job's list and, if it detects that a file is already in a list, it will not be put in again, NOR WILL IT MOVE TO THE FIRST POSITION.

There is one general list and as many job lists as there are jobs that opened files, or even more than that if lists were LOADed.

The general list is immediately adjacent to the thing itself. Its size is of:

rcnt_end + xx * rc_entryl bytes (see dev8_keys_recent_thing) where xx is the number of files configured by the user. The general list always exists.

Job lists are in heaps, i.e. memory allocated on the common heap, one per job that opens a file. The memory is allocated for job 0 and doesn't go away when the job is removed. Heap size is:

rcnt_hdr + xx * rc_entryl bytes (see dev8_keys_recent_thing) again, xx is the number of files as configured by the user.

23.1.2. Job IDs

The primary way of identifying which list goes with which program is, of course, the job Id. Each list contains the ID of the job that opened it, and when the same job tries to open a new file, its name is added to the list of the job with the same Job ID. However, as a general rule, when trying to work with the RECENT Thing and it requires a Job ID, it is better to pass it a job name:

Over the course of a session, a user will typically launch several programs which will open a variety of files. Many of these programs will really be transient and will be removed from the system when they are done (e.g. a quick file search with FiFi). (Note jobs started from Hotkeys are also created and removed, just as if they were loaded from disk).h en a program is removed, its list stays in the system, for several reasons.

First it is to speed up the system - constantly removing and adding new lists (which are allocated on the common heap) would just slow the system down (see performance penalty, below).

Second, the information should be saved when the RECENT Thing lists are saved to disk, to be re-loaded in a later session.

Third, and most importantly, a mechanism is provided for jobs with the same name but different job Ids to use only one list.

As an example, think of assembling all of the SMSQ/E sources. During such a run, the assembler gets called hundreds of times (once for each file to assemble) : this means that the assembler program is executed (and then removed after assembling one file) hundreds of time. Each time it is executed, it gets a new Job ID. So when it tries to open its first file, the system won't find any list for it, since it has an all-new Job Id. If that happens, the RECENT Thing tries to find a list for a job with the same name as the job calling it. If it finds such a list, it changes the Job Id to the one calling it and adds the file to that list. (The search for a list with the same name is made with a hash of the name).

Note that this scheme only works with jobs that keep the same name. QD, for example, changes its name when you load a file, and Xchange also changes its name sometimes. In that case, if the job must be found via a hash, it will not be possible to find the previous list.

The same problem also arises when LOADing the lists at boot time (or whenever): The lists are stored with the Job IDs as they were when the list was saved.

In a nutshell, when trying to use the RECENT Thing, it is better to pass it a job name rather that a Job ID. The mechanism to do that is explained below (see JobIDs and Name pointer for the INFO extension).

It is important to understand that, if two jobs have the same name, they will be using the same list (e.g. several instances of SBasic, unless you changed their name).

23.1.3. Buffers

Whenever a buffer is required, this means an area of memory starting at an even address. The thing does not check that the address is even, if it isn't, mayhem may ensue.

23.2. The Thing interface in Assembler

The RECENT Thing is built as an extension thing. As usual, the name of the extension should be contained in D2 when trying to USE the thing. When calling the extension, A1 points to the parameter block/list/stack. Parameters are passed as is, without interspersing them with **thp.xxx** parameter qualifiers. If a call requires a parameter, that parameter is always compulsory.

Job IDs may always be passed as -1 to denote the calling job. However, see the ***** JobIDs and Name pointer note under the "INFO" extension.

Example for using GFFJ extension of the thing

```
; A0        s
; A1  c     s     Points to a suitable buffer to hold filename
; A2-A4     s

; D0  r     error
; D1  c     s     The job ID (maybe -1)
; D2-D4     s
; D5  c     s     The size of the buffer

got_id
      move.l      d1,d4             ; keep jobID
      move.l      a1,a4             ; and buffer pointer
      sub.l       a5,a5             ; no name pointer *******
      move.l      #'GFFJ',d2        ; extension to use
      lea         rcnt_name,a0      ; point to name of thing
      moveq       #-1,d3            ; wait forever
      moveq       #-1,d1            ; I will use the thing
      moveq       #sms.uthg,d0      ; use thing
      jsr         gu_thjmp          ; on return A2= ptr to thg header, a1 to thg
      tst.l            d0           ; ok?
      bne.s       gt_exit           ; no!
      move.l      a1,a0             ; pointer to thing
      sub.l       #14,sp            ; get some space for parameters
      move.l      sp,a1             ; and point to it
      move.w      d5,(a1)           ; buffer size
      movem.l     d4/a4/a5,2(a1)    ; insert buffer, jobID, name pointer
gt_cont
      jsr         thh_code(a0)      ; call extn thing
      add.l       #14,sp            ; reset stack
      move.l      d0,d5             ; remember error
      lea         rcnt_name,a0      ; free thing
      moveq       #sms.fthg,d0
      moveq       #-1,d1
      jsr         gu_thjmp
      move.l      d5,d0             ; restore error
gt_exit           rts

rcnt_name
      dc.w  6,'Recent'
```

******* Please see the explanation of the name pointer below.

23.2.1. JobIDs and Name Pointer

With the exception of the ADDF extension, all of the extensions that use job Ids as parameter use the following scheme

1 - If the job ID is a "genuine" Job ID, that ID is used to try to identify the list for that job. A "genuine" job ID is a Job ID for a job that actually exists currently in the system. If no list can be found for this job, the thing tries to find the name for this job and get the list with that name. As usual, -1 is a genuine job ID and denotes the current job.

2 - if the job ID is passed as -2, then the next long word on the parameter stack is expected to be a long pointer to the name of the job. This allows you to search for lists for jobs that are no longer executing (and for which, thus, no genuine job ID exists any more). If there is no name, set a long word 0.

For the ADDF extension, a genuine Job ID is always necessary (-1 is accepted).

23.2.2. The extensions

The RECENT Thing provides the following extensions:

INFO An extension to get some information on a list.

JOBS An extension to get some info on the jobs the thing holds lists for.

ADDF An extension to add a file to the list. Should not be used.

GFFA An extension to get the first (=most recent) file name in the general list.

GFFJ An extension to get the first (=most recent) file name in the list for a certain job.

GALL An extension to get all filenames from the general list.

GALJ An extension to get all filenames from the list for a certain job.

GARR An extension to get all filenames from the general list into an array.

GARJ An extension to get all filenames from the list for a certain job into an array.

SAVE An extension to save the lists to a preconfigured file.

LOAD An extension to load the lists from a preconfigured file.

REMV An extension to remove a list from the thing.

SYNC An extension to synchronise job IDs with those in the lists.

INFO

Get INFO on list

Call parameters		Return parameters	
D1		D1	maximum number of files per list
D2		D2	High word : max str length (NOT incl. length word) Low word : nbr of strings (see below)
D3		D3	pointer to heap space IF one was looking for a job list, not the general list
A1	Parameter list	A1	Preserved
A2	Thing linkage block	A2	Preserved

Error returns

IJOB there is no list for this job

Parameter list (pointed to by A1)

JobID	Long	use -2 if you want to check with the name
Name pointer	Long	pointer to job name or 0 if none

This gets some information about the list of files maintained by the thing.

If the Job ID is 0, this will be the general files list, if not, it will be the list for that job. For a jobID of -1 and -2, see “JobIDs and Name Pointer” section above.

On return, A2 contains the buffer size for a "GALL" or a "GAFJ" extension call. The size is the size necessary to store all strings + a length word for each string + a possible byte necessary to even out each individual string length.

D1 contains, on return, a word with the maximum number of files in any list. All lists have the same capacity. It is configured by the user.

On return, the high word of D2 contains the length of the longest filename currently being stored by the Recent list. This length may vary, though, if a new file with a longer name is later opened. It will never exceed 41 characters, though (but you should think of the necessary length word).

The number of files in the list, returned in the low word of D2, may also vary, if a new file is later opened. However, it will never exceed the maximum number of files as configured.

If no file is yet contained in the list, D2 will be 0 (this should normally not happen).

JOBS

Get a list of jobs

Call parameters		Return parameters	
D0		D0	Special return, see below
A1	Parameter list	A1	Preserved
A2	Thing linkage block	A2	Preserved

Error returns: see below

BUFL the buffer was too short. In this case as much as possible is filled in

Parameter list (pointed to by A1)

Length of buffer Long

Pointer to buffer Long

This gets a listing of all jobs for which the thing holds lists of files.

For each job, the listing in the buffer holds the Job ID in a long word followed by a standard string with the job name (which my be 0 if the job has no name).

The list is terminated by a long word of -1. If the buffer is too small to hold all job names, the buffer will be filled in as much as possible, and the error err.bufl will be returned.

There is no guarantee that any of the jobIDs returned are still valid : if the job with that ID has been removed, the jobID will no longer be valid.

The return in D0 is special : unless it is a negative error code, it holds the number of jobs in the buffer.

ADDF

Add a file

Call parameters		Return parameters	
D2		D2	???
A1	Parameter list	A1	Preserved
A2	Thing linkage block	A2	Preserved

Error returns:

IJOB	Invalid job ID

Parameter list (pointed to by A1)

JobID	Long	The ID of the job to add the file for
Ptr to string	Long	Filename, normal string

This adds a file to the lists. Normally, a program should not call this routine. The adding of files is handled by the system whenever a file is opened.

!!!! Use this extension at your own risk !!!!

The jobID must be a genuine ID of a job actually executing.

NOTE: this extension goes into supervisor mode to avoid the list being modified by several jobs at once.

GFFA

Get First File for Any job

Call parameters		Return parameters	
A1	Parameter list	A1	Preserved
A2	Thing linkage block	A2	Preserved

Error returns:

BFFL	The buffer for the return string is too small - in this case nothing is copied to the buffer.

Parameter list (pointed to by A1)

Buffer length	Word	word with maximum length of buffer
Ptr to buffer	Long	

Gets the first (i.e. most recently opened) file from the general list. If the buffer is too small to contain the filename, nothing will be copied to the buffer.

Filenames will never exceed 41 bytes (+2 bytes for the length word).

GFFJ

Get First file For Job

Call parameters		Return parameters	
A1	Parameter list	A1	Preserved
A2	Thing linkage block	A2	Preserved

Error returns:

BFFL	The buffer for the return string is too small - in this case nothing is copied to the buffer.

Parameter list (pointed to by A1)

Buffer length	Word	(signed!) word with max length of buffer
Ptr to buffer	Long	
Job ID	Long	should be -2 if a name is supplied
Name pointer	Long	pointer to job name if jobID = -2, else 0

Gets the first (i.e. most recently opened) file for the job passed as parameter. For a jobID of -1 and -2, see "JobIDs and Name Pointer" section above.

If the buffer is too small to contain the filename, nothing will be copied to the buffer.

Filenames will never exceed 41 bytes (+ 2 bytes for the length word, which the buffer should provide for). If the list is empty, this will be a null length string. The list may be empty if there is no list for this job.

GALL

Get ALL files

Call parameters		Return parameters	
A1	Parameter list	A1	Preserved
A2	Thing linkage block	A2	Preserved

Error returns:

BFFL	if the buffer was too short, in this case as much as is possible is filled in

Parameter list (pointed to by A1)

Buffer length	Long	This is a long word
Ptr to buffer	Long	

This gets all filenames from the general list into a buffer. The filenames will be copied one after the other, the name of the most recently opened file being the first one to be copied. If the filenames don't all fit, as many as possible will be copied, and error **err.bffl is** returned.

Filenames are typical SMSQ/E strings, with a length word in front and evened out to start at an even address.

The buffer length is passed as a long word. The buffer must start at even an address.

GALJ

Get ALI files for Job

Call parameters		Return parameters	
A1	Parameter list	A1	Preserved
A2	Thing linkage block	A2	Preserved

Error returns:

JOB	no list could be found for a job with that ID or name
BFFL	the buffer was too short, in this case as much as is possible is filled in

Parameter list (pointed to by A1)

Buffer length	Long	This is a long word
Ptr to buffer	Long	
Job ID	Long	should be -2 if a name is supplied
Name pointer	Long	pointer to job name if jobID = -2, else 0

This gets all filenames from the list for a job into a buffer. The filenames will be copied one after the other, the name of the most recently opened file being the first one to be copied. If the filenames don't all fit, as many as possible will be copied, and error **err.bufl** is returned.

Filenames are typical SMSQ/E strings, with a length word in front and evened out to start at an even address. The list is ended with a 0 length word.

The buffer length is passed as a long word. The buffer must start at even an address.

For a jobID of -1 and -2, see “JobIDs and Name Pointer” section above.

GARR

Get all files into an ARRay

Call parameters		Return parameters	
A1	Parameter list	A1	Preserved
A2	Thing linkage block	A2	Preserved

Error returns:

IPAR	The array had a null dimension or the length of the elements in the array is smaller than the maximum length of a string in the list

Parameter list (pointed to by A1)

Ptr to array	Long	The pointer to array is absolute, it is NOT relative to A6
Array size	Word	Number of elements in array
Elemt length	Word	Length of elements in array, INCLUDING length word

This is similar to the GALL call, in that all, or as many as possible, filenames will be copied. Here however, they will be evenly spaced. If the filenames don't all fit, as many as possible will be copied, and no error is returned.

Filenames are typical SMSQ/E strings, with a length word in front and evened out to start at an even address.

The array is just a buffer into which the filenames will be evenly spaced. It can thus be viewed as a two-dimensional string array.

If there is still some space left between one filename and the next, the remainder will be filled with 0.

GARJ

Get all files into an ARay, for a Job

Call parameters		Return parameters	
A1	Parameter list	A1	Preserved
A2	Thing linkage block	A2	Preserved

Error returns:

IPAR	The array had a null dimension or the length of the elements in the array is smaller than the maximum length of a string in the list

Parameter list (pointed to by A1)

Ptr to array	Long	The pointer to array is absolute, it is NOT relative to A6
Array size	Word	Number of elements in array
Elemt length	Word	Length of elements in array, INCLUDING length word
JobID	Long	Job ID should be -2 if a name is supplied
Name pointer	Long	name pointer pointer to job name if jobID = -2, else 0

This is similar to the GALJ call, in that all, or as many as possible, filenames will be copied. Here however, they will be evenly spaced. If the filenames do not all fit, as many as possible will be copied, and **no error** is returned.

Filenames are typical SMSQ/E strings, with a length word in front and evened out to start at an even address. The array is just a buffer into which the filenames will be evenly spaced. It can thus be viewed as a two-dimensional string array. If there is still some space left between one filename and the next, the remainder will be filled with 0.

For a jobID of -1 and -2, see "JobIDs and Name Pointer" section above.

SAVE

SAVE all lists to configured file

Call parameters		Return parameters	
A2	Thing linkage block	A2	Preserved

Error returns:

	Any error from file operations
FDNF	No file is configured

This saves the lists for all jobs currently held in the thing, into the file configured by the user. The file is overwritten. You can't specify another file.

The general list is NOT saved. The name of the SAVE file is NOT added to any list of files, not even the general one, when SAVEing or LOADing.

LOAD

LOAD lists from configured file

Call parameters		Return parameters	
A2	Thing linkage block	A2	Preserved

Error returns:

INAM	The file was not a valid RECENT Thing save file
	Any error from memory allocation/deallocation
	Any error from file operations

This loads the lists as saved by the SAVE extension. All lists existing in the thing, except for the general list, will be removed prior to loading. The file from which the lists are loaded is as configured by the user, you can't specify another file.

Thus, if you LOAD the lists as the very first thing in your boot file, they will also fill up with the files opened up during your normal boot.

If you LOAD the lists at the end of your boot file, they will replace all lists generated up to that time (except for the general list).

When SAVEing or LOADing, the name of the SAVE file is NOT added to any list of files, not even the general one.

It is possible to save the lists, re-configure SMSQ/E to use lists with a different size, and load the saved lists after a reboot with the newly configured SMSQ/E. In that case:

- If the new list size is smaller than the saved size, only some files will be copied to the new lists. THERE IS NO GUARANTEE THAT THESE will include the newest files opened.
- If the list size is larger than the saved size, all filenames will be copied.

In the latter case, and also when the sizes stay the same between saving and loading, the order of the filenames will be preserved.

It is recommended to use the SYNC extension after loading the lists (see below).

REMV

REMoVe a list for a job

Call parameters		Return parameters	
A1	Parameter list	A1	Preserved
A2	Thing linkage block	A2	Preserved

Error returns:

IJOB	There is no list for this job
	Any error from memory allocation

Parameter list (pointed to by A1)

JobID	Long	Job ID should be -2 if a name is supplied
Name pointer	Long	name pointer pointer to job name if jobID = -2, else 0

This removes the list for the job passed as parameter.

For a jobID of -1 and -2, see “JobIDs and Name Pointer” section above.

SYNC

Tries to give current Job IDs to jobs in the list

Call parameters		Return parameters	
A2	Thing linkage block	A2	Preserved

Error returns:

OK unless error in job information trap

As explained above, the Job IDs stored in the RECENT Thing may not correspond to the Job IDs of the jobs currently executing, for example after loading the lists. The SYNC runs through the list of all iobs currently executing in the system and, if a list exists for a job with that name, it sets the Job ID of that list to that name.

23.3. SBasic keywords

The following keywords allow use of the thing from SBasic:

RCNT_INFO	A function to get some information on a list.
RCNT_JOBS	A function to get some info on the jobs the thing holds lists for.
RCNT_ADDF	A keyword to add a file to the list. Should not be used.
RCNT_GFFA$	A function to get the first (=most recent) file name in the general list.
RCNT_GFFJ$	A function to get the first (=most recent) file name in the list for a certain job.
RCNT_GALL	A function to get all filenames from the general list.
RCNT_GALJ	A function to get all filenames from the list for a certain job.
RCNT_GARR	A keyword to get all filenames from the general list into an array.
RCNT_GARJ	A keyword to get all filenames from the list for a certain job into an array.
RCNT_SAVE	A keyword to save the lists to a preconfigured file
RCNT_LOAD	A keyword to load the lists from a preconfigured file
RCNT_REMV	A keyword to remove a list from the thing.
RCNT_SYNC	A keyword to synchronize job ids with those in the lists.
RCNT_HASH$	A function to get a hash from a string.

length = **RCNT_INFO** (*[job_id,] str_nbr%,str_len%,max_nbr%*)

Get information on a list of files

length = space needed for getting all strings, including length word

ob_id = optional id of job the info is about

EITHER as a long int where

0 means get the general list,

-1 means get the list for myself (= default if omitted)

OR as a string with the name of the job

str_len% = RETURN parameter, max length of string

str_nbr% = RETURN parameter, number of strings currently held

max_nbr% = RETURN parameter, max number of strings in lists

This gets some information about the lists of files maintained by the thing, either the general list (job_id) = 0 or the list for a certain job. The Job ID may be passed as a long word, or, preferably, as a string with the entire name of the job.

On return the function returns the size for using the "**RCNT_GALL**" or "**RCNT_GALJ**" keywords. The size is the size necessary to store all strings + a length word for each string + a possible byte necessary to even out each individual string length.

The other three parameters are filled in on return of the function:

- The str_len% parameter contains the length of the longest filename currently being stored by the Recent list for this job, on in the general list. This length may vary, though, if a new file with a longer name is later opened. It will never exceed 41 characters, though.

- The number of files in the list, returned in the str_nbr% parameter, may also vary, if a new file is later opened.

- Finally, the max_nbr% is the maximum number of files a list may hold (as configured by the user).

result% = **RCNT_JOBS** (*length,buffer*)

Get a list of all jobs into a buffer:

length = length of buffer, in bytes

buffer = space for the list, preferrably a space allocated with ALCHP

result% = 0 or +ive: number of jobs in the list

else negative error code (e.g. "buffer full" if the buffer was too small)

-1 means get the list for myself (= default if omitted)

if there is an error, as much as possible is filled in the buffer

This gets a listing of all jobs for which the thing holds lists of files.

For each job, the listing in the buffer holds the Job ID in a long word followed by a standard string with the job name (which my be 0 if the job has no name).

The list is terminated by a long word of -1. If the buffer is too small to hold all job names, the buffer will be filled in as much as possible, and the error **err.bufl** will be returned.

There is no guarantee that any of the jobIDs returned are still valid : if the job with that ID has been removed, the jobID will no longer be valid.

The return in value holds the number of jobs in the buffer, unless it is a negative error code.

RCNT_ADDF *[job_ID,] filename$*

adds a file name to the list

filename is the name of the file to add

job_id is the optional jobID of the job supposed to have opened the file

(defaults to -1, i.e. myself)

This adds a file to the list.

USE OF THIS KEYWORD IS STRONGLY DISCOURAGED.

Normally, a program should not call this, the adding of file is handled by the system whenever a file is opened.

The JobID MUST be passed as a long word.

file$ = **RCNT_GFFA$** ()

gets the first (=most recent) file name in the list.

If the list is empty, this will be a null length string.

Returns the name of the first (i.e. most recently opened) file from the general list. If the list is empty, this will be a null length string.

file$ = **RCNT_GFFJ$** (*job_ID*)

gets the first (=most recent) file for the given job - if none found, this is an empty string.

job_id is optional, if not given, search for current job

Gets the name of the first (i.e. most recently opened) file for the job passed as parameter. If the parameter is omitted, it will default to -1, i.e. the current job. If the list is empty, the result will be a null length string.

result% = **RCNT_GALL** (*length,buffer*)

Get ALL file names from the general list into a buffer.

length = length of buffer - this should be at least as much as that returned by the RCNT_INFO keyword

buffer = space for list

result% = 0 if all went ok else negative error code:

err.bffl	Buffer too small
err.ipar	Wrong number of parameters
err.ijob	Wrong job id
	Any error from the thing use routine

If the error is err.bffl, as much as possible is filled in the buffer

This gets all filenames of the general list into a buffer. The filenames will be copied one after the other, the name of the most recently opened file being the first one to be copied. If the filenames don't all fit, as many as possible will be copied and the error "buffer full" is returned.

Filenames are typical SMSQ/E strings, with a length word in front and evened out to start at an even address. This might be used as follows:

```
str_len%=0
str_nbr%=0
str_max%=0
blength=RCNT_INFO(,0,str_nbr%,str_len%,str_max%): rem get info on size
buffer=ALCHP(blength) : rem get buffer
result%=RCNT_GALL (blength,buffer)
```

Perhaps a better way to get the filenames for the basic programmer is the **RCNT_GARR** function.

result% = **RCNT_GALJ** ([*jobID,] length, buffer*)

Get ALL file names for a job into a buffer.

length = length of buffer - this should be at least as much as returned by the RCNT_INFO keyword for this jobs

buffer = space for list

job_id = (optional) id of job:

EITHER as a long int where -1 means get the list for myself (=default)

OR as a string with the name of the job

result% = 0 if all went ok

else negative error code:

err.bffl buffer too small

err.ipar wrong number of parameters

err.ijob wrong Job ID

any error from the thing use routine

if the error is err.bffl, as much as possible is filled in the buffer

This gets all filenames of the list for a job into a buffer.

The filenames will be copied one after the other, the name of the most recently opened file being the first one to be copied. If the filenames do not all fit, as many as possible will be copied and the error "buffer full" is returned.

Filenames are typical SMSQ/E strings, with a length word in front and evened out to start at an even address.

This might be used as follows:

```
str_len%=0
str_nbr%=0
str_max%=0
blength=RCNT_INFO("Prowess",str_nbr%,str_len%,str_max%): rem get info on size
buffer=ALCHP(blength) : rem get buffer
result%=RCNT_GALL ("Prowess", blength,buffer) : rem data info buffer
```

RCNT_GARR *array$*

Get all filenames from the general list into an ARRay

array$=a 2 dimensional string array.

This is similar to the GALL call, in that all, or as many as possible, filenameswill be copied. Here however, they will be copied into what must be a two-dimensional string array (i.e. DIM a$(xx,yy). If the filenames do not all fit, as many as possible will be copied, and no error is returned.

An error bad parameter will however be returned if:
- The array isn't a two-dimensional string array
- The second dimension of the array is too small for the longest element in the list.

Note that filenames will not be longer than 41 characters, so a DIM a$(x,41) will guarantee that that error won't happen.

The first array element to be filled in will be element 0.

This might be used as follows:

```
str_len%=0
str_nbr%=0
blength=RCNT_INFO(0,str_nbr%,str_len%,str_max%) : rem get info on size
DIM files$(str_nbr%,str_len%): rem or better dim files$(str_nbr%,41)
RCNT_GARR files$
```

RCNT_GARJ *[jobID,] array$*

Get all filenames for a job into an ARRay

job_id = (optional) id of job:
EITHER as a long int where -1 means get the list for myself (=default)
OR as a string with the name of the job
array$=a 2 dimensional string array.

This is similar to the GALJ call, in that all, or as many as possible, filenames will be copied. Here however, they will be copied into what must be a two-dimensional string array (i.e. DIM a$(xx,yy). If the filenames don't all fit, as many as possible will be copied, and no error is returned.

An error bad parameter will however be returned if :

- The array isn't a two-dimensional string array
- The second dimension of the array is too small for the longest element in the list.

  Note that filenames will not be longer than 41 characters, so a DIM a$(x,41) will guarantee that that error won't happen.

The first array element to be filled in will be element 0.

This might be used as follows:

```
str_len%=0
str_nbr%=0
blength=RCNT_INFO("Prowess",str_nbr%,str_len%,str_max%) : rem get info on size
DIM files$(str_nbr%,str_len%)  : rem or better dim files$(str_nbr%,41)
RCNT_GARJ "Prowess",files$
```

hash$=**RCNT_HASH$**(*string$*)

Returns the hash from the string

string$ = a normal string to turn into a hash

This returns the hash, as used by the RECENT Thing for job names, from the string passed as parameter. The hash algorithm used is a very simple one, the consideration was speed over anything else. So this hash is certainly very easy to break and probably not very collision proof....

That said, running it over an entire qxl.win file with about 10.000 files did not give any collision for any of the filenames.

RCNT_SAVE

SAVE lists to configured file : no parameters

This saves the lists for all jobs currently held in the thing, into the file configured by the user. The file is overwritten. You can't specify another file.

The general list is NOT saved. The name of the SAVE file is NOT added to any list of files, not even the general one, when SAVEing or LOADing.

RCNT_LOAD

LOAD lists from configured file : no parameters 1

This loads the lists as saved by the RCNT_SAVE extension. All lists existing in the thing, except for the general list, will be removed prior to loading. The file from which the lists are loaded is as configured by the user, you can not specify another file.

Thus, if you LOAD the lists as the very first thing in your boot file, they will also fill up with the files opened up during your normal boot.

If you LOAD the lists at the end of your boot file, they will replace all lists generated up to that time (except for the general list).

When SAVEing or LOADing, the name of the SAVE file is NOT added to any list of files, not even the general one.

It is possible to save the lists, re-configure SMSQ/E to use lists with a different size, and load the lists after a reboot with the newly configured SMSQ/E. In that case:

- If the new list size is smaller than the saved size, only some files will be copied to the new list. THERE IS NO GUARANTEE THAT THESE will include the newest files opened.
- If the list size is larger than the saved size, all filenames will be copied.

In the latter case, and also when the sizes stay the same between saving and loading, the order of the filenames will be preserved.

RCNT_REMV [*jobid*]

REMoVe a list for a job

This removes the list for the job passed as parameter. If no such job exists, it returns an error.

RCNT_SYNC

Tries to give current Job IDs to jobs in heap

As explained above, the Job IDs stored in the RECENT Things may not correspond to the Job IDs of the jobs currently executing, for example after loading the lists. The SYNC runs through the list of all iobs currently executing in the system and if a list exists for a job with that name, it sets the Job ID of that list to that name.

23.4. Configuration

Using the usual standard config program, you can configure:

- Whether SMSQ/E should use the RECENT Thing at all. If not, neither the thing nor the SBasic extensions for it will be initialised/usable.
- The size of the lists (i.e. how many files they should contain):

  The maximum allowed size is 255. The minimum allowed size is 1.
  The longer the list, the higher the performance penalty (see below).
  All lists have the same size. The default list size is 20.

- The name of the SAVE file if you want to be able to save/load the lists between sessions.

23.5. Performance penalty

There is, of course, a performance penalty involved when opening files, since the RECENT Thing must be used and the lists searched through, but the time necessary to check and add the file to the list is small.

As an indication, compiling all of SMSQ/E, under SMSQmulator, in a version of SMSQ/E 3.23 without the RECENT Thing takes about 118 seconds. The same with the RECENT Thing takes about 125 seconds. This is with a list size of 250 files.

Under QPC, these were 58 seconds without the RECENT Thing and 63 seconds with the Recent thing.

24. Appendix A Compiling SMSQE with SMSQEMake

24.1. Compiling the source code

The SMSQ/E sources can be compiled either the "easy way" or the "hard way". The easy way is to use the program called "extras_exe_SMSQEMake" which will be described here.

If you want to do this the "hard way" (why?), there is a document called "HowTo" in the subdirectory "extras_help" in the SMSQ/E sources. Please note that this document is now outdated and will not be kept up to date.

SMSQEMake is intended to make compiling and linking the sources for SMSQ/E easier. It is a replacement for the "make_bas" and "flp_bas" files found in the smsq_ directory for each of the targets.

For this, you are presented with the names of the module (link) files. You then select the ones you want to have recompiled. The very short version of using the program, if you are impatient and the sources are contained in a qxl.win container, is:

- EXEC "dev8_extras_SMSQEMaket". Remember : the sources for SMSQ/E are supposed to lie on device “dev8_’ and SMSQEMake expects them to be..
- Select the target(s) you want: at least the Generic + the machine for which you want to compile SMSQ/E.
- You are presented with the names of the module (link) files. You then select the ones you want to have (re-) compiled, or select “All” to compile all modules.
- Select "Make".
- Hit "OK" and wait until compilation is complete. A warning window pops up if there were errors during compilation.

Please note that, if you want to use the "drive" item, this software needs the menu extensions. Normally there should be no need to use this item, hence the menu extensions are not needed. There are also some other requirements if you want to recompile the SMSQEMake program itself (see below).

SMSQEMake does presume that the sources are on one drive and in the usual directories. The drive MUST be called "dev8_": all of the sources presume that they are on a drive called dev8_ . So you need the dev device which is already contained in SMSQ/E. Apart from the menu extensions, all other necessary extensions are already contained in the compiled version of the program. Moreover, if the complete qxl.win container with the sources has been downloaded, all ancillary programs (assembler, linker etc...) are also on the “disk”.

So, unless you want to see what other programs are required, you could now just skip right to the next section (see How to use.. below).

24.2. Requirements

The requirements for a successful compilation of SMSQ/E are that you will need a "dev" device, an assembler, a linker, the "make" program, plus a concatenator. These are all supplied together with the sources, the DEV device can be found in SMSQ/E itself.

24.2.1. The DEV device

The source files are set up in such a way that they expect to be found on a device called "DEV8_".

DEV is the usual SMSQ/E "dev" device which can refer to any actual physical device, you should use DEV_USE 8, xxx to set this up. Thus, if the source files are or, say, win6, use:

DEV_USE 8, win6_

To allow easy recompilation on all sorts of systems, all references to include and other files must be made to files located on device 'dev8_'.

24.2.2. The assembler

You will also need an assembler to compile the individual source files into what are know as "_rel" files, i.e. relocatable compiled files, which are later bound together by a linker. For the time being, the assembler must be the QMAC Assembler. This is available in the dev8_extras_exe_ directory.

24.2.3. The linker, cctf and make programs and how to use them.

On the source device, in the "exe" subdirectory, you will find the Linker, cctf, cct and Make programs. It is recommended, but not strictly necessary, to load the assembler, the linker and the make programs as resident programs via hotkeys, such as:

```
ert hot_res('z',<device>_make)
ert hot_remv('z')
ert hot_res ('z',<device>_linker)
ert hot_remv('z')
ert hot_res ('z',<device>_QMAC)
```

You don't HAVE to do this, but the Make program always attempts to execute the Assembler and Linker from an executable Thing rather than an executable file first. If you do not do that, you must make sure that these programs can be found in the "PROG_USE" path. If they can not be found as executable things, nor in the PROG_USE path, the make program will not be able to find them and will crash.

The cctf program should lie in the normal PROG_USE path.

Here is a short description of what each of these programs does, though it is not necessary to know this to use SMSQEMake, which will drive the programs normally.

24.2.3.1. The Make Program

That program takes a simple linker command file (_link) and checks which files have to be re-assembled, it then assembles them, and causes the linker to be executed.

The presumes that your computer keeps a correct time and date, since the files' timestamps are used by the make program to check whether they need to be re-compiled or not.

The Make program also presumes that you have a program called QMAC to compile all files that need to be compiled, a piece of software called Linker to link them and a program called cctf to concatenate libraries.

The Make program s a compiled S*Basic Program. The source of that can be found in the "exe" directory under the "source" sub-directory which can modified to as required.

A special toolkit is required, called “OUTPTR_BIN” which can also be found under the "source" subdirectory.

24.2.3.2. The linker

The linker is responsible for linking the “_rel” files generated by the assembler. It will generally be called from the Make Program and should be loaded as mentioned above.

24.2.3.3. CCTF

The purpose of the cctf program is to concatenate “_rel” files into libraries. It should simply be put into your normal PROG_USE path. If you want to use it by hand, use it as follows:

```
EX <device>_cctf,cct$,lib$
```

where cct$ is a file containing only the names of the rel files to be concatenated.

There is also a small cct program which you will need for the QXL. This can be found in the exe directory and should be put into your Program path.

LRESPR a file called "dev8_extras_cline_bin” in job 0.

24.3. How to use SMSQEMake

24.3.1. Setting up the environment

To make things easier, you can use a small boot program called dev8_extras_compile_boot. This will load the linker, assembler and make programs as resident things (to make access faster) and LRESPR “cline_bin” and will then execute the SMSQEMake program.

Line 100 of this boot program MUST be amended to point to the correct device for win2_

Otherwise, just exec the SMSQEmake program. It opens on its main screen. You can also pass the program certain command line parameters.

24.3.2. Description of the program

24.3.2.1. The title bar

In the upper title bar, you can see the usual ESC, move, size, sleep items. You can also see the DEL, SaT Dir, ?, OK , All and Make,items, which are explained below.

In the title bar there is also a drive item where you can chose the drive on which the link files can be found. In this version of SMSQEMake, all link files must be on the same drive.

MOST IF NOT ALL SOURCE AND LINK FILES PRESUME THAT THE FILES TO BE ASSEMBLED RESIDE ON DEV8_.

24.3.2.2. The targets row

Under the title bar, there is a row with many buttons. They select/deselect the "targets" for which you want to (re)compile the sources. The names of the targets should be self-explanatory.

24.3.2.3. The link files window

Underneath this is the application subwindow with the names the "link" files where, generally, each link file represents one module. As targets are de/selected, the relevant link files for this target are added to, or deleted from, this window (not from the disk, of course).

24.3.2.4. The "All" item

The 'All' item de/selects all of the link files. Any of them can be de/selected individually, by clicking on them.

24.3.2.5. The "OK" item

When the "OK" item is hit/done, the program starts building the targets.

24.3.2.6. The DEL item

If the DEL item is selected, then before building the targets is started, SMSQEMake will try to delete all previously compiled files, i.e. all “_rel”, “_err”, “_map_”, “_log” and “_lib” files (but not, of course, the source files themselves!) and the compiled targets. This is done by "EXECing" the file called "dev8_extras_ del_all_bas". Note that EXECing a basic file is a feature of SMSQ/E and may not work under different OSes

Please also note:

For this to work, you must LRESPR a file called "dev8_extras_cline_bin" into job 0. If you don't, the program will not work. If you run the boot program in job 0, this is done automatically.

24.3.2.7. The "Make" item

There is also a "Make" item. If you select that before compilation starts, then if there was no error during compilation of the source file(s) for a certain target, the resulting target file will actually be built. You can also simply rebuild a target by selecting the target and deselecting all link files for it, if the MAKE button is selected.

The targets made will be called as follows, <dev> being the device with the sources (I presume dev8_):

Generic:	no target file per se.
Atari:	<dev>smsq_atari_smsq.prg
(Super)GoldCard:	<dev>smsq_gold_gold
	<dev>smsq_gold_gold8
QPC:	<dev>smsq_qpc_smsqe.bin
Qx0:	<dev>smsq_q40_rom
QXL:	<dev>smsq_qxl_smsqe.exe
SMSQmulator:	<dev>smsq_java_java
Aurora:	<dev>smsq_aurora_smsqe
ptr_gen / wman:	<dev>ee_ptr_gen
	<dev>ee_wman_wman
Q68:	<dev>smsq_q68_smsq_4_win
Q-emulator:	<dev>smsq_qem_smsq_qem

24.3.2.8. The “SaT” item

The item selects all targets. There is no corresponding unset, you must deselect the selected targets individually.

24.3.3. Command line parameters

When executing SMSQEmake, you can pass it some command line switches. They are all two letters long and are preceded by a hyphen. They are:

A - General:

Name	**Meaning**
-qa	Quit after program finishes
-q0	Quit if 0 errors
-as	Autostart: start compiling as soon as program is launched
-mk	"Make" item is selected
-de	Delete all rel, lib, module and target files before compiling
-sa	The “All” item is selected: select all modules
-fa	Force assembly of all files
-ta	Select all targets

B - Target selection
Then there are command line switches to select individual targets (but not modules):

Name	Target
-tg	Generic SMSQE - routines used by nearly all targets
-tc	QPC
-tq	Q40
-ti	Atari
-to	(Super) Gold card
-tu	Aurora
-tx	QXL
-t8	Q68
-tr	PtrGen
-tm	Q-emulator
-tj	SMSQmulator

24.3.4. A proposed way of working

The program can be used as follows:

- LRESPR a file called "dev8_extras_cline_bin” in job 0 if you want to use the DEL item.
- LOAD the dev8_extras_compile_boot Program. This sets up the dev8_ device to point to win2_ Remember, if your device with the sources is NOT win2_ change it in line 100.
- EX the SMSQEMake program.
- Select your target(s).
- Select the module (link) file(s) to be recompiled, possibly with the All item.
- Set the "Make" item so that a final version will be built.
- Start everything with "OK".

The program now treats every module that is selected. If the process for a module did not result in an error (i.e. assembling and linking went OK), then the module is made available again.

24.3.5. Error reports

Error report facilities in this program are, as yet, very rudimentary:

If the entire process went OK, the program has made all modules available again. If not, a small window pops up telling you that there was an error (it doesn't tell you where the error was, though - just look which module(s) remained selected, then look in the source code at the log file for each module).

Thus, at the end of a compile run, you can see whether all module files compiled OK or not - if they stay selected, there was an error.

24.4. Recompiling or changing SMSQEMake

The source for SMSQEMake is in the extras_exe_source directory.

To compile it, you will need the dev8_extras_source_SMSQEMake_bin file (containing the compilation options), the “outptr_bin” file (in the dev8_extras_exe_source directory) and the QPTR extensions.

All of these files must be LRESPR'd before compilation.

24.5. Additional programs

New subdirectories have been created as compared to the original source code that came from Tony Tebby, they are all located in one subdirectory called “extras”.

Directory	Contents
extras	Additional documents and programs, mostly those used by myself in an attempt to make compilation easier. Any program put into that directory should have a corresponding help

file in the help directory.

This has several new subdirectories itself:

Sub Directory	Contents
_help	Help files are there to explain how some programs work. Not how new SMSQ/E features work.
_new	Documentation on features that are new to THIS version of SMSQ/E. In the next version, this documentation is moved to either the _doc or the _help subdirectories.
_exe	Executable programs.
_doc	Documentation on new features.
_html	Programs generating some of the html files for the SMSQE website.
_source	The source files for the assembler and some basic programs found in the dev8_extras_ and dev8_extras_exe_ subdirectories.

25. Appendix B Official SMSQ/E style guide

The purpose of this document is to keep a single coding style within the whole project. SMSQ/E has evolved over more than a decade with many people involved, therefore most but not all existing source files comply with the style described in this guide. New source files however have to comply. If changes in existing files are done that don't comply with the style described here it is your choice to either adapt to the style of the given file or make your changes in the style described in this guide.

25.1. Generic requirements

25.1.1. Development system

The standard distribution is assembled using the QMAC, QLINK and QMAKE assembler tools. All submissions must be compatible with these tools, the only exception being hardware dependant code which may need other tools for certain purposes (e.g. 68020+ assembler commands).

25.1.2. Assembler

All parts not specific to a certain hardware must be written in plain 68000 language and must be compatible with the QMAC assembler syntax.

25.1.3. Character set

The normal QDOS/SMSQ character set is to be used.

25.1.4. TAB stops

TAB limits need to be set to 8 characters. You are encouraged to configure your editor to compress multiple spaces to TAB characters in order to keep files small.

25.2. Assembler files

25.2.1. Generic file structure

A source file starts with one or more header lines followed by a changes list, the "section" command, xdefs, xrefs, include files and finally the code itself.

Example:

```
; Routines to do something brilliant V1.02 (C) 2002 Fred Flintstone
; Barney Rumble
; Addition information about the file (optional)
; 2002-01-01      1.00  First release (FF)
; 2002-06-20      1.01  Added br_evenbetter function (BR)
; 2002-12-31      1.02  Fixed serious buffer overflow in sub-function
;

      section     brilliant

      xdef  br_super
      xdef  br_evenbetter

      xref  cv_ctype

      include     'dev3_keys_sms_io'

      [Code]

      end
```

The header line gives a short explanation of the purpose of the file, the current version number and the list of names of people that hold the copyright (additional names are added in an extra line).

The changes list is something new and so far you won't encounter it in any original files of the distribution. As the code is available to all developers, keeping a detailed track of the changes within a file becomes a must. When working on a file, always increase the version number and write down your changes in the list!

The format of the list is

```
;YYYY-MM-DD v.vv    Description of change (full or abbreviated name of author)
```

The date is in ISO8601 notation (big words for "year first, month next and day last).

25.2.2. Headers

A header within a source file always starts with the line

```
;+++
and ends with the line
;---
```

All routines available to external files should have a function header.

Function headers contain a description of the function followed by a detailed in/out table as shown here:

```
;+++
; XYZ driver - read function (one line description, optional)
;
; This function does nothing really, it is just an example for a function
; header (detailed discussion of function)
;
;     d0    cr          drive number / error status
;     d1    r           byte read
;     d2          s
;     a3    c     p     linkage block
;     a5    c     u     pointer to something
;
;     status return standard
;---
```

The register list is a sorted list of all registers affected (D0 first, A6 last).

Following the register name is a standard field with several characters that define the function and behaviour of the register:

```
       d0    xy    z     description
```

x: " " or "c" = "call parameter"
y: " " or "r" = "return parameter"
z: " ", or "p" = "preserved", "u" = "updated" or "s" = "smashed"

So in the example header above D0 is a call parameter with the drive number and also a return parameter with the error status. D1 is a return parameter, too, whereas D2 is just smashed. A3 is a call parameter and is preserved. A5 is a call parameter that gets updated within the call.

Registers not listed must be preserved if not stated otherwise.

After the register field the status of the flags after the call is documented.

This can be for example:

- "status return standard"
- "status return arbitrary"

25.2.3. Cases

Labels and the assembler mnemonics themselves are lower-case. Comments and headers are either lower or mixed case.

25.2.4. Comments

Comments are started using the ";" operator and must be written in English.

25.2.5. Labels

Labels are lower case, words are separated using the "_" character. Normally a label has its own line.

The labels belonging to the body of a function usually start with a short hand for the function name, i.e.

```
com_check   ->   comc_loop
com_rxen    ->   crxe_iact
com_iserve  ->   comi_error
...
```

25.2.6. References to include and other files

All new references to include and other files must be made to files located on device 'dev8_'

Use dev_use 8,xxxy_ to set the dev device to whatever you want.