Appendix K : NATIONAL VARIANT DIFFERENCES

This appendix notes the differences between the various versions of Kernel produced for national variants.

K.1 North American Variant

K.1.1 APPENDIX E

Character $23 is # (hash), not £ (pound).

K.1.2 APPNDIX F - Table F.1

Entry should read

$CA	202	£	(pound sign)

K.1.3 APPENDIX G - Table G.1

Entries should read:

$23	35	#	# or 3.s	(hash)
$CA	202	£	3.A	(pound)
$D3	211			(not used)
$EO	224	recall	recall or recall.s	(labelled "flash")

K.1.4 APPENDIX H

Entries should read:

# £ 3	33 3	23 #	CA £	1E rs

RECALL	EO recall	EO recall	////	////

PRNT #	23 #	D8 print	DB printa	CF printm

0. DOCUMENT CONTROL

0.1 Contents List

0.5 Document Cross References

1. PSD 76.97.7 OPD System Overview
2. PSD 76.97.1 OPD Hardware Specification
3. PSD 76.97.4 OPD Telephony Module
4. PSD 76.97.6 OPD ULA3 Specification
5. R51002 OPD Handbook
6. PSD 76.97.13 OPD Establishment Checks
7. PSD 76.97.3.2 OPD Director Facilities for Application Writers
8. PSD 76.97.8 OPD Printer Functional Specification
9. PRESTEL Terminal Specification (Edition One, The Post Office)
10. TMS 5220A Voice Synthesis Processor Data Manual (Texas Instruments, Preliminary Printing DM-02, October 1982)
11. TMS 6100 Voice Synthesis Memory Data Manual (Texas Instruments, Tentative Data, June 1980).
12. MC68000 16/32-Bit Microprocessor; Programmer's Reference Manual (Fourth Edition)
13. MC68008 16-Bit Microprocessor with 8-Bit Data Bus. Motorola Semiconductors.
14. PSD 76.97.3.3 OPD Telephone Handler Interfaces for Application Writers
15. OPD/ADM/2 Publishing Procedures for OPD Base Functional Software (internal working document)
16. OPD/SPEC/3 OPD Printer Manager Specification

1.3 Documentation Conventions

Where instruction mnemonics and hardware register names are mentioned without furthur explanation, these refer to Motorola MC68008 features documented in the manufacturer's literature [ref 12,13].

A labelled box such as

DO | a | b | c | d |

indicates the contents of the specified register or store location, where 'a' is the most significant byte (bits 2^{24} to 2^{31}) and 'd' the least significant (bits 2^{0} to 2^{7}).

This document adopts the Motorola convention of prefixing hexadecimal numbers with a dollar sign.

Most definitions are shown in terms of Assembler 'names' rather than numbers. The exact mapping of such names onto numbers is deliberately left undefined in this document. Anyone writing software to this Kernel specification should consult the relevant issue documentation and use the correct symbol definitions and values for that issue.

Where definitions are given in terms of absolute numbers they are not expected to change, but it is suggested that names are also used for these items.

1.4 Availability of Facilities

This document describes the complete set of facilities currently implemented for Release 1 of the OPD system software. Various releases of Kernel apply to the OPD Release 1 software - consult the software release notice applicable to that release to determine the availability of any given facility. See also Appendix K of this document.

b) the ability for a new or different application to gain immediate access to the screen and keyboard

c) the ability to partition the physical keyboard into more than one 'logical' keyboard, and to alter that partitioning dynamically

d) the necessity for certain keys to cause high level interrupts that affect the entire operation of the OPD System

2.4 Software Structure

2.4.1 ACTIVITIES AND SCHEDULING

In principle, several different things can be happening simultaneously on the OPD. To enable the implementation of such facilities, the system runs as a number of independent activities (tasks/jobs/programs/processes or whatever). Each activity will normally be responsible for carrying out a fairly self-contained function, (which may or may not be directly apparent to the user). The activities are each run under the control of a multi-tasking scheduler in Kernel, which also handles the low level device interrupts.

In general, by making the relevant requests to Kernel, an activity may:-

a) allocate and use various I/O devices

b) allocate memory space on a permanent or temporary basis

c) create and destroy other activities

d) coordinate itself and communicate with other activities

e) suspend itself and/or other activities until various events occur

2.4.2 INTEGRITY OF THE USER INTERFACE

To preserve the integrity of the user interface to the telephone system, screen and special system function keys at all times, the activities in the OPD system will be expected to conform to a number of fairly stringent constraints regarding their behaviour in certain circumstances.

These constraints are imposed partly by Director routines and partly by coding conventions.

Where activities obey code that has not been written to the relevant coding standards (such as user written assembly language programs), the activity obeying such code is expected to be associated with another activity which operates in the role of a user code guardian. This latter activity exercises a degree of

3. INTERFACES TO KERNEL

3.1 General Comments

Kernel is the lowest level of the overall OPD 'Operating System', and as such provides interfaces to the hardware and software features in a number of different ways. Most of the application software and the remainder of the system software use the normal Kernel call mechanism described below to access the facilities specified in the bulk of this document.

There are also a number of other interfaces that are outlined in this Chapter.

3.2 Interrupt and Exception Handling

The MC68008 exception vector is located at address zero [ref 12,13]. On the OPD, this address occurs in the system Read Only Memory area, and consequently all initial processing of exceptions is of necessity performed by the ROM based software (i.e. Kernel).

To enable Kernel's handling of certain interrupts and exceptions to be bypassed, some of these are vectored indirectly through RAM. For hardware test programs and the like, a system call (see section 4.2.2) is provided to allow one or more exceptions to be vectored other than into Kernel. Note that this facility is not intended for general software use.

A different mechanism is provided for program error and TRAP exceptions, enabling them to be used as a means of entering routines in user mode (as opposed to supervisor mode). This mechanism operates on an activity by activity basis, allowing partial or complete bypassing of the OPD system software handling of such traps. This mechanism is intended to allow both high level language program error recovery and the ability to emulate different Operating Systems by enveloping System Calls.

3.3 Initial Entry

On the occurrence of a power up reset, Kernel runs a number of establishment checks [ref 6], and assuming that these are passed, then the various system tables and devices are initialised, and a known single activity is started.

Further details of the initialisation performed on the devices, and the interface to the system's initial activity will be found in the relevant sections of this document, and are summarised in Chapter 20.

3.4 System Errors

Following the detection (or the deliberate forcing) of a system error, the initial entry sequence (including the establishment

instruction had been obeyed immediately prior to re-entering the activity at the instruction following the TRAP.

The error codes are all referred to by names of the form

ERR.cc

where the two character tag indicates the actual error detected. This two character tag is used in the remainder of the document.

A combined summary of all the possible error codes in the OPD base software is given in Appendix B.

Some calls return further information in DO on a successful return; in such cases the value of DO.L will be positive or zero.

On error returns, the whole of DO.L is set negative, but all the error codes are in the range -1 to -127. This means that to check for a particular error, only the least significant byte of DO need be compared, after establishing that DO.L is negative.

3.6.2 BAD PARAMETER ERRORS

Most of the Kernel calls that report individual error codes in circumstances specified in the call descriptions can also report error BP - bad parameter - in circumstances that are not always described explicitly. Such an error arises from general checking that Kernel performs on parameters passed to the call, and indicates that one or more parameters is not just invalid, but garbaceous.

For example, if rubbish is passed to a call that expects an activity identifier, then the call will probably return error BP. If the rubbish happens to look like a genuine activity identifier but is not, or if a genuine activity identifier is passed that is otherwise invalid for some reason, then a return will be made with error NA - invalid activity identifier.

d) Spurious Interrupt

A system error is forced.

e) Level 2 Autovector
f) Level 5 Autovector

These exceptions enter the relevant Kernel interrupt handler routines.

g) Level 7 Autovector

A system error is forced.

4.2.2 EXCEPTION AND INTERRUPT REDIRECTION KERNEL CALL

Trap Name: T.KERNEL
Action Value (DO.B): K.XXXVECT

Additional Call Parameters:

D1.W : exception vector code number
AO : address of exception routine

Return Parameter:

A1 : address of previous exception routine

Error Return:

BP : bad parameter

This Kernel call enables the RAM held indirection table of exception vectors to be altered.

The alterable exception vectors are coded in D1 as follows:-

0	Bus Error
1	Address Error
2	Illegal Instruction
3	Spurious Interrupt
4	Level 2 Autovector
5	Level 5 Autovector
6	Level 7 Autovector

Entry to the nominated exception routine is in Supervisor mode, as if the hardware exception vector pointed directly at the routine, but obviously there is a small overhead in execution speed.

Note that following the use of this Kernel call, some or indeed all of the facilities described in this document may be rendered inaccessible or otherwise unusable. The previous exception

Following this call, Kernel resets the hardware, preserves the four byte identifier in non-volatile memory, and enters the Establishment Check routine as though a power-up reset had occurred. The preserved four byte identifier is then passed to the initial Director activity when the system is restarted, (see section 20.3).

4.4 Address Space Query Kernel Call

Trap Name: T.KERNEL
Action Value (DO.B): K.CHECKADD

Additional Call Parameter:

AO	:	start of area to be checked.
A1	:	buffer address
D1.W	:	length of buffer

Error Returns:

BP : bad parameter (address does not exist)

This call enables an activity to determine whether a given area currently exists within the address space of the OPD, and if it does, to read the contents. This call is not for general use; the specification of this call will change in future releases of the OPD software.

The contents of the area are copied, a byte at a time, into the specified buffer. If the specified area does not exist, the hardware generates a Bus Error, which will result in Kernel returning to the activity with error BP.

Reading a non-existent address stops the bus for a significant period of time, so this call should be used in moderation.

5.4 Activity Types

There are four types (ranks) of activity handled by Kernel, as described below. They are ranked in decreasing order of privilege.

5.4.1 SPECIAL KERNEL ACTIVITIES

These activities are used by Kernel for special purposes, they have special capabilities, and are not discussed any further in this document.

5.4.2 SYSTEM ACTIVITIES

These are activities that provide the overall control of the OPD System, in terms of high level scheduling, telephone management, and response to the special key interrupts. These activities may allocate and use devices not directly available to user applications.

5.4.3 TRUSTED ACTIVITIES

These are activities that obey application code written to the relevant standards and/or Director interfaces, and are trusted to respond to the high level user interactions in the correct manner.

5.4.4 UNTRUSTED ACTIVITIES

These are activities that may possibly obey code not written to the relevant standards and/or Director interfaces. Each such activity is controlled (loosely speaking) by a trusted or system activity acting as a user code guardian. Untrusted activities have reduced capabilities in some circumstances.

5.5 Activity Hierarchy

The various activities running in the OPD system have a hierarchical ordering which is maintained by Kernel, and which can be interrogated to some extent.

In general, activities may create and manipulate other activities of equal or lesser rank. When a new activity is created, Kernel remembers which activity performed the creation. When an activity creates another one of lower rank, the creator is also registered as the guardian of the new activity. When an activity creates another of equal rank, the creator's guardian is registered as the guardian of the new one.

Under normal circumstances, activities registered as the guardian of one or more lower ranked activities should not terminate before the lower ranked activities, and Kernel takes steps to prevent this unless such action is deliberately requested.

illegal in some manner, and is waiting for attention from its guardian

5.8 Scheduling

All activities that are 'running' are given a share of the CPU by the scheduler. The actual share each activity receives is governed by the activity's priority, which is a number in the range 1 to 127. A priority of zero puts the activity into the suspended/inactive state. Note that it should not be assumed that the scheduler actually provides 127 different levels of priority discrimination.

5.9 Error Trapping

Each activity has an associated exception vector which controls what happens when that activity obeys an instruction which causes an exception. More than one activity can share the same activity exception vector, and a default one is supplied when the activity is set up.

The format of an activity exception vector is a word aligned table containing 26 four byte entries. Each entry either contains zero (meaning that the Kernel supplied default action for that exception is required), or it contains a user supplied routine address. On the occurrence of the relevant exception, the user supplied routine will be entered as described below.

The ordering of the entries in the activity exception vector is as follows:

0. Bus Error
1. Address Error
2. Illegal Instruction
3. Zero Divide
4. CHK Instruction
5. TRAPV Instruction
6. Privilege Violation
7. Trace
8. Line 1010 Emulation
9. Line 1111 Emulation
10. Trap 0
11. Trap 1
...
25. Trap 15

If a user supplied activity exception vector is held in RAM, it may be updated dynamically, as desired.

When a user supplied routine is entered via the activity exception vector, the effect is the same as if entry were via the MC68008 exception vector, except that the processor will be in User Mode, and the stacked information will be dumped on the activity's stack. The user supplied routine can hence return to

At bootstrap time only one activity is created and started by Kernel. Any other activities must be created by the OPD system software by using this Kernel call.

The type of the new activity must be of equal or lesser rank than that of the caller, and is encoded in D1.W as follows:-

4 Special Kernel Activity
3 System Activity
2 Trusted Activity
1 Untrusted Activity

The guardian of the new activity will be the caller when creating an activity of lesser rank, and the caller's guardian when creating one of equal rank.

If the initial priority is non-zero, the new activity is scheduled for running. If the initial priority is zero the new activity will not run until its priority is increased by the caller (or some other activity). When first run, the activity will start at the initial PC address, with the stack and registers set up as described in Section 5.6. Note that both the initial PC and register dump area pointer must be word addresses.

If either the initial PC address or the dump area address falls within the allocatable RAM area, then the actions outlined below are taken for each address. (Chapter 8 describes the Memory Management system and terminology.)

The actions taken when an allocatable RAM address is given are :-

a) if the address is not within an allocated segment, or the segment is a cell allocator segment, then error NS - invalid segment - is returned

b) if the segment is transient, then the ownership of the segment is changed to the new activity

c) if the segment is mobile, it is marked as frozen by the new activity, and immobile

If the rank of the new activity is higher than that of the calling (creating) activity, then error NA - invalid activity rank - is returned.

If there is no room in memory or Kernel's tables for the new activity's control block, then error OM - out of memory - is returned.

5.10.2 DESTROY ACTIVITY

Trap Name: T.KERNEL
Action Value (D0.B): K.XACT

Error Returns:

none (no return is made)

This call destroys the calling activity. If the calling activity is guardian to other activities, those subsidiary activities are also destroyed, even if they are active. This course of action may prove unwise.

Segments, semaphores and I/O channels are treated in the same manner as for DESTROY ACTIVITY above.

5.10.4 FORCIBLY DESTROY ACTIVITY (MURDER)

Trap Name:	T.KERNEL
Action Value (D0.B):	K.MURDER

Additional Call Parameter:

D1.L : target activity identifier

Error Returns:

NA	:	invalid activity identifier
BP	:	bad parameter

This call forces the target activity to 'commit suicide' irrespective of its current state, or of the state of its subsidiaries. The calling activity must either be the guardian of the target activity or share the same guardian activity. This call should be used with caution. For the effect on the target activity, see above.

5.10.5 NOTIFY ACTIVITY EXCEPTION VECTOR

Trap Name:	T.KERNEL
Action Value (D0.B):	K.EXVECT

Additional Call Parameter:

A0 : new activity exception vector address

Error Return:

BP : bad parameter

This call notifies the address of a new exception vector for the current activity. For further details see section 5.9.

If the address specified is zero, the default handling of all exception and error traps is reinstated for the current activity.

Additional Call Parameters

D1.L	:	target activity identifier (or -1)
D2.W	:	priority value (0 to 127)

Error Returns:

NA	:	invalid activity identifier
NC	:	not complete (holding semaphore(s))
BP	:	bad parameter

This call enables an activity to change the priority of another activity.

If the target activity identifier is given as -1, the calling activity's priority will be changed.

If the target activity is holding one or more semaphores, the priority remains unchanged and return made with error NC - operation not complete (activity holds semaphores).

Note that a priority of zero will put the target activity into the suspended/inactive state. If such a call is made when the target activity is waiting for an event or a semaphore, then it will be suspended, but its Program Counter will be stepped back so that the Kernel Call which caused the wait will be re-issued when the activity is subsequently allowed to run.

If the calling activity sets its own priority to zero, it will not be re-entered again without the intervention of another activity.

5.10.8 GIVE ACTIVITY'S SUBSIDIARIES

Trap Name:	T.KERNEL
Action Value (DO.B):	K.GIVSUBSID

Additional Call Parameters:

D1.L	:	target activity identifier (or -1)
D2.W	:	buffer length in long words
AO	:	buffer address

Return Parameter:

DO.W	:	number of entries in buffer

Error Returns:

NA	:	invalid target activity
BO	:	buffer overflow
BP	:	bad parameter

6. EVENT SYSTEM

6.1 General Comments

Kernel provides a rudimentary event system that enables activities to wait for one or more events to occur, or determine whether they have occurred. An activity may also cause events to happen. As well as possessing an event identity (number), an event is also classified in two different ways, namely that it can be a Kernel event or an Internal event, and a Local or a Global Event.

6.2 Kernel Events and Internal Events

Kernel events are those that result from a Kernel detected occurrence of some kind, and have a defined meaning in Kernel.

Internal events are those which are of significance only to the underlying OPD system software, and have no inherent meaning ascribed to them by Kernel. Such events can only occur at the express request of an activity, by making a call on Kernel.

6.3 Global and Local Events

Certain events are of interest to a number of activities at the same time. These events have a unique event number by which they are known to all activities, and are known as global events. The occurrence of a global event is notified simultaneously to all the activities that have requested such notification.

Events that are of interest to only one activity, (or one activity and others that communicate with it) are known as local events. Different activities may ascribe different meanings to a given local event number. In general, local events are only notified to a given single activity. An example is the event that can be generated by request when input becomes available on an I/O channels.

6.4 Event Numbers

Each event has a number in the range 0 to 31. Each activity has two 'registers' handled by the event system, and both of these are 32 bits long.

The bits in these event system registers correspond to event numbers on a one-to-one basis, such that bit 2^n in the registers represents event number n.

Where a Kernel Call may potentially be concerned with more than one event at any given time, then the events concerned are passed to the call as a long word, with individual bits set for those events that concern the particular call. This mechanism applies to all the calls described in this chapter.

6.7 Waiting and Timeouts

Kernel calls that involve a potential wait (namely WAIT FOR EVENT and LOCK SEMAPHORE) specify the maximum duration of the wait in D2.

A negative value specifies that the wait is indefinite (and hence the activity will not continue until the item waited for actually occurs).

A zero value specifies that no wait at all is required or permitted, and the call will return immediately, either with a succesful return, or with error NC - operation not complete (timed out).

A positive value specifies the maximum duration of wait that is permitted, in display frame interrupts, i.e. in units of approximately 20ms. The value specified should make due allowance for any margin of error induced by frame interrupts lost during microdrive activity. If the expected action has not occurred within this period, return is made with error NC - operation not complete/timed out.

Activities using the application screen or playing a vital role in preserving the integrity of the User Interface should use indefinite or long waits with extreme caution.

6.8 Event System Kernel Calls

6.8.1 REQUEST FUTURE EVENT NOTIFICATION

Trap Name:	T.KERNEL
Action Value (DO.B):	K.ASKEVENT

Additional Call Parameter:

D1.L : event flags (see section 6.4)

Error Returns:

none

This call requests the (future) notification of one or more events. The bits specified in D1.L correspond to the bits in the activity's Event Request Register, with the relevant bits being set for those events of which the activity requires notification (see Appendix C).

The event flags in D1 are added (logical OR) into the caller's Event Request Register.

Error Returns:

NA	:	invalid activity identifier
BP	:	bad parameter

This call signals to a specified activity that one or more internal local events have occurred. The bits in D1.L correspond to the internal local events in the Event Request Register, with a bit set for each event the activity wishes to cause, (see Appendix C). The local event(s) specified are signalled by Kernel to the indicated activity, which will receive notification in due course if it has been requested.

If the target activity does not exist, return is made with error NA - invalid activity. If it exists, but has not requested notification then the event is effectively lost.

6.8.5 WAIT FOR EVENT(S) AND SPECIFIED PERIOD

Trap name:	T.KERNEL
Action Value (D0.B):	K.WAIT

Additional Call Parameters:

D1.L	:	event notification subset (see section 6.4)
D2.W	:	wait time/timeout period (see section 6.7)

Return Parameter:

D1.L	:	notified events

Error Return:

NC	:	operation not complete (timed out)

This call enables an activity to wait for a specified period of time, and on the same Kernel call receive notification of any events previously requested.

The call allows for a subset of notifiable events to be handled on a given call by passing a subset mask in D1. Only those events for which the relevant bits are set in D1 will be handled on a particular call.

A timeout value is specified in the manner described in section 6.7, where the wait specified may be indefinite, or a given (possibly zero) period.

If, at the time the call is made or at any time during the specified wait period, notifiable events occur or have already occurred (i.e. bits get set in the activity's Event Notification Register), the call immediately makes a successful return. On such a return, D1 is set to the logical AND of the activity's Notification Register and the value of D1 when the call was made.

7. SEMAPHORE SYSTEM

7.1 General Comments

In any multi-tasking system there is a need to synchronise cooperating activities, and allow mutually exclusive access to shared data. While it is possible to achieve such effects without any special primitives, on a single processor system there is an implied interaction with the scheduling mechanisms.

Consequently, Kernel provides some mutual exclusion primitives, and use of them (as opposed to any other) is obligatory. The Kernel primitives allow the creation and manipulation of simple semaphores (sometimes known as locks).

A treatise on the use and abuse of semaphores and the mutual exclusion problem is beyond the scope of this document; it is widely discussed in computing literature.

Two general rules which are worth repeating are

a) a semaphore should preferably be held for as short a time as possible (i.e. just when the protected information/state is inconsistent)

b) if two or more semaphores are required to be held by an activity at any given time, then these semaphores must always be obtained in the same order

A specific rule of the OPD system is that an activity using the application screen should not normally wait indefinitely for a lock, as it must be prepared to relinquish the screen at short notice.

Note that Kernel does not itself associate a given semaphore with any particular resource; this is a function of the higher level software.

7.2 Semaphore Identifiers

Each semaphore in the system has a unique identifier, which is used whenever a reference to that semaphore is required.

When a semaphore is created, the identifier is returned by Kernel to the creator, whereupon it may be used by any activity requiring access to the semaphore.

Physically, a semaphore identifier consists of a long word with a unique positive and non-zero value in the top word, and a semaphore number in the bottom word. The unique values are allocated sequentially by Kernel. The semaphore number is an index into Kernel's tables.

7.4.2 DESTROY SEMAPHORE

Trap Name: T.KERNEL
Action Value (DO.B): K.XSEM
Additional Call Parameter:

D1.L : semaphore identifier

Error Returns:

NL	:	invalid semaphore identifier
IU	:	semaphore in use
BP	:	bad parameter

This call destroys a previously created semaphore.

If the semaphore identifier is invalid, return is made immediately with error NL - invalid semaphore identifier.

If the semaphore is currently locked to the destroying activity, it is first unlocked (and possibly then locked to another).

If the semaphore is transient, and the calling activity is not the owner of the semaphore, then return will be made with error IU - semaphore in use.

If any other activity is now holding the semaphore (i.e. has it locked), the semaphore is not destroyed, and return is made with error IU - semaphore in use. In such cases, then if the semaphore is transient then the activity that has locked the semaphore will be chosen as the new owner.

Following a successful return from this call, the semaphore identifier becomes invalid for further use and any such attempt will return error NL.

7.4.3 CHANGE SEMAPHORE OWNERSHIP

Trap Name: T.KERNEL
Action Value (DO.B): K.SEMOWNER

Additional Call Parameters:

D1.L	:	semaphore identifier
D2.L	:	target activity identifier

Error Returns:

NA	:	invalid activity identifier
NL	:	invalid semaphore identifier
BP	:	bad parameter

This call changes the ownership of a specified semaphore. If the semaphore does not exist, then return is made with error NL -

Error Returns:

NC : not complete (timed out)
UV : unexpected event
NL : invalid lock identity
BP : bad parameter

This call enables an activity to obtain/lock a semaphore and at the same time respond to any events that might occur before the lock is obtained. If the semaphore is available (unlocked), then it is locked to the calling activity, which is re-entered immediately (scheduling constraints permitting).

If the semaphore is locked to another activity, then the calling activity is made to wait as specified by the timeout parameter (as for a WAIT FOR EVENT call, see section 6.7). If at the start of the wait, or at any time during the wait, an event gets set in the calling activity's Notification Register, then return is made with error UV - unexpected event occurred - and a WAIT FOR EVENT call can be made to determine the event(s). If no events occur and the lock fails to become available to the calling activity within the specified timeout period, return is made with error NC - operation not complete/timed out.

Activities waiting for a given semaphore are given the lock and allowed to proceed in strict order of the 'lock' requests.

[Note: a request to lock a semaphore already held by the calling activity is handled as if the lock were available].

7.4.6 UNLOCK SEMAPHORE

Trap Name: T.KERNEL
Action Value (DO.B): K.UNLOCK

Additional Call Parameter:

D1.L : semaphore identity

Error Return:

NL : invalid semaphore identifier
BP : bad parameter

This call unlocks a semaphore currently held by the activity. The next activity waiting for the semaphore (if any) will be given it, and allowed to proceed.

If the indicated semaphore is not currently held by the calling activity, or the semaphore does not exist, then return is made with error NL - invalid semaphore/lock.

segment address (the segment identifier), which can be remembered while the segment is not actively in use.

Kernel does not restrict segment access to any particular activity (although many will in fact be private), and free access to segments by any activity is allowed. Any contention over the sharing of information in the segments is the responsibility of the activities concerned.

To aid multiple access to a segment, Kernel maintains a list of which activities are actively using (freezing) a particular segment at any given time; this information controls whether a segment and/or its contents may be deleted, or moved.

8.3.1 SEGMENT IDENTIFIERS

Each segment in the system has a unique identifier, which is used whenever a reference to that segment is required.

When a segment is created, a segment identifier is returned by Kernel to the creator, whereupon it may be used by any activity requiring access to the segment.

Physically, a segment identifier consists of a long word with a unique positive and non-zero value in the top word, and a segment number in the bottom word. The unique values are allocated sequentially by Kernel. The segment number is an index into Kernel's tables.

8.3.2 TRANSIENT AND PERMANENT SEGMENTS

Each segment in the system is either a transient or a permanent segment. A transient segment is regarded as being 'owned' by a particular activity, while a permanent segment has no such ownership.

The ownership of a segment is not used to control access to a segment in any way; the **only** difference is that if an activity is destroyed, then any transient segment owned by it will also be destroyed and the space returned to the free store area (unless some other activity is using it). A permanent segment remains until explicitly destroyed.

Note that the creation and deletion of permanent segments should normally be performed via the relevant Director interfaces [ref 7].

8.3.3 SEGMENT PLACEMENT AND MOBILITY

When a segment is created, some control over its placement may be effected by defining the expected usage. If a segment is expected to exist for a long time, but only accessed occasionally, e.g. the telephone directory, then it should be created as a 'mobile' segment. Such segments are allocated at as

Normal segments are expanded and contracted from the top (high address) end; cell allocator segments from the bottom (low address) end.

Mobile cell allocator segments are expanded and contracted automatically, whenever the cell allocation scheme sees fit; the size of a normal segment must be changed explicitly.

A limit can be placed on the maximum and minimum sizes of a segment, in which case any attempt to extend or contract beyond the limits will be prevented.

8.4 Cell Allocation

The space in a normal segment may be freely used by any activity accessing it. Alternatively, a segment may be created so that the space is further divided up into variable length record-like units known as cells with formal mechanisms for their manipulation. The cell allocation facilities are closely related to the segment allocation facilities, and there is a certain amount of interaction between the two.

In general, cells within a segment are freely movable, and may be expanded, contracted or deleted at will. This means that care is needed to ensure that cells are not moved while their addresses are remembered.

Kernel provides a relocatable form of cell address, which can be remembered while the cell is not actively in use, and should be used whenever it is not strictly necessary to use the actual address of a cell.

8.4.1 CELL TAGS

Each cell in the system has a unique name or 'tag', which is used whenever a reference to that cell is required.

When a cell is created, a cell identifier (tag) is returned by Kernel to the creator, whereupon it must be used whenever access to the cell is required. The segment in which the cell resides is implicitly defined by the cell tag.

Physically, a cell tag consists of a word containing a tag number, which is an index into the cell allocator's tables and is always non-zero. A zero tag may be used with safety to indicate 'no cell'.

There can be a maximum of (64K - 1) cells in existence at any one time across all the cell allocation segments in the OPD system.

The process of converting from a cell tag to a real address is quick and efficient.

Note carefully that a cell tag is still a form of RAM address,

The 'first' cell is defined as the cell having the highest address within the segment. This will be the first cell created in the segment, unless that cell is deleted, or changed in size. In this latter case, the cell which becomes 'first' is indeterminate, and may change with time.

8.5 Dynamic RAM Management Kernel Calls

8.5.1 CREATE NEW SEGMENT

Trap Name: T.KERNEL
Action Value (D0.B): K.NEWSEG

Additional Call Parameter:

D1.L	:	target activity identifier (or -1)
D2.W	:	initial segment size
D3.W	:	placement control flags (see below)

Return Parameter:

D0.L	:	new segment identifier

Error Returns:

NS	:	no space left in segment table
NA	:	invalid activity identifier
OM	:	out of memory
BP	:	bad parameter

This call creates a new transient segment with the specified characteristics, and returns the identifier of the segment.

The target activity is registered as the owner of the segment. If the target activity identifier is passed as -1, the calling activity will be so registered.

The initial size of the segment is specified in units of 512 bytes; zero length segments cannot be created, and such an attempt will return with error BP - bad parameter.

If it is not possible to create a segment of the required size, return is made with error OM - out of memory. If there is no free left in Kernel's segment table, return is made with error NS entry - no space in segment table.

The placement control flags specify the type and placement characteristics of the segment, as follows:-

bit 2 : 0 => normal segment
        1 => cell allocator segment

bit 3 : 0 => segment is mobile

relevant Director facility [ref 7].

8.5.3 CHANGE SEGMENT OWNERSHIP

Trap Name:	T.KERNEL
Action Value (DO.B):	K.SEGOWNER

Additional Call Parameter:

D1.L	:	segment identifier
D2.L	:	target activity identifier (or -1, or 0)

Error Returns:

NA	:	invalid activity identifier
NS	:	invalid segment identifier
BP	:	bad parameter

This call changes the ownership of a specified segment. If the segment does not exist, then return is made with error NS - invalid segment identifier.

The new owner activity is passed in D2.L. If this parameter is passed as zero, then no activity will regarded as the new owner and the segment will be made permanent. If the parameter is non-zero, the specified activity will be regarded as the new owner of the segment, and the segment will be made transient. If the value -1 is specified, the calling activity will be the new owner.

Note that the ownership of segments should normally be changed via the relevant Director facility [ref 7].

8.5.4 SET SEGMENT LIMITS

Trap Name:	T.KERNEL
Action Value (DO.B):	K.SEGLIMIT

Additional Call Parameters:

D1.L	:	segment identifier
D2.W	:	maximum permitted size
D3.W	:	minimum permitted size

Error Returns:

NS	:	invalid segment identifier
BP	:	bad parameter

This call informs the segment allocation system that the specified segment should not be permitted to extend or contract beyond the given limits. If the segment does not exist, return is made with error NS - invalid segment identifier.

The maximum and mimimum permitted sizes of the segment are given

Note carefully that an attempt to expand a frozen segment will first of all thaw the segment as far as the calling activity is concerned. If the segment is frozen by other activities, then no expansion takes place and error IU - segment in use - is returned. An attempt to expand a segment will potentially change the address of **any** thawed segment, and hence the addresses of any cells in such segments. You have been warned !

8.5.6 REDUCE SEGMENT

Trap Name:	T.KERNEL
Action Value (D0.B):	K.SEGREDUCE

Additional Call Parameters:

D1.L	:	segment identifier
D2.W	:	number of blocks to reduce by

Error Returns:

NS	:	invalid segment identifier
IU	:	segment in use
OM	:	out of memory
BP	:	bad parameter

This call reduces the segment by a given amount. If the segment does not exist, return is made with error NS - invalid segment identifier.

The amount of the required reduction is given in units of 512 bytes and must be positive.

Note that normal segments are reduced from the top (high address) end, while cell allocator segments are reduced from the bottom (low address) end.

Note carefully that an attempt to reduce a frozen segment will first of all thaw the segment as far as the calling activity is concerned. If the segment is frozen by other activities, no contraction takes place, and error IU - segment in use - is returned.

If the segment is used by the cell allocator, and the existing cells cannot be squashed (compacted) into the remaining space, no change is made to the segment size, and error OM - out of memory - is returned.

Cell addresses within the segment are changed by this call.

Immobile segments may be reduced if desired but cannot subsequently be expanded back to their original size.

Please purchase Image2PDF on http://www.verypdf.com/ to remove this message.

If an immobile cell allocation segment is thawed, the segment itself will not move, but any cells within it become movable.

If the segment is not actually frozen by the calling activity, then no further action is taken, and a successful return is made.

8.5.9 GIVE SEGMENT INFORMATION

Trap Name: T.KERNEL
Action Value (DO.B): K.GIVSEG

Addtional Call Parameter:

D1.L : target segment identifier

Return Parameters:

DO.L : segment information (see below
D1.L : segment identifier (see below)
D2.L : owning activity identifier
D3.L : segment limit information

Error Returns:

NS : invalid segment identifier
BP : bad parameter (see below)

This call enables certain information about a segment to be determined. If the segment does not exist, return is made with error NS - invalid segment identifier - and the remaining registers are unaltered.

DO returns the following information:

DO	s	s	f	a

s : size of the segment (in units of 512 bytes)

f : further information -

bit 0 : 0 => segment is thawed (a=0)
1 => segment is frozen (a≠0)

bit 1 : 0 => segment is transient
1 => segment is permanent

bit 2 : 0 => normal segment
1 => cell allocator segment

bit 3 : 0 => segment is mobile
1 => segment is immobile

a : number of activities freezing the segment

Because of store fragmentation, intervening frozen segments and/or the effects of multi-tasking, it will not necessarily be possible to create a segment of this size, or extend an existing segment by this amount.

The call also returns the total number of 512 byte units available to the memory management scheme, in D1.W.

8.5.11 CREATE NEW CELL

Trap Name:	T.KERNEL
Action Value (D0.B):	K.NEWCELL

Additional Call Parameter:

D1.L	:	segment identifier
D2.W	:	number of bytes required

Return Parameter:

D0.W	:	new cell tag (in range 1 to 64K-1)

Error Returns:

NS	:	invalid segment identifier
IU	:	segment in use (frozen)
OM	:	out of memory
BP	:	bad parameter

This call creates a new cell within the specified segment, and returns the tag of the cell. The tag may be used to obtain the actual address of the cell.

If the segment does not exist, return is made with error NS - invalid segment identifier .

The actual size of the cell may be up to 5 bytes larger than the requested size.

Note that this call first thaws the target segment as far as the calling activity is concerned (enabling it to be moved unless it is immobile). It potentially changes the addresses of all movable, thawed segments and hence the cells in them, and in particular the addresses of other cells in the segment in which the new one is being created, **even if that segment is immobile.**

If the segment remains frozen by any activities other than the calling activity, the call returns with error IU - segment in use - and the cell will not be created.

If there is insufficient free space obtainable in the segment even after squashing all the existing cells together, an attempt to extend the segment is made by a call on EXPAND SEGMENT (see

Note that this call first thaws the target segment as far as the calling activity is concerned (enabling it to be moved unless it is immobile). It potentially changes the addresses of all movable, thawed segments and hence the cells in them, and in particular the addresses of this and all other cells in the segment in which the cell is being expanded **even if that segment is immobile.**

If the segment remains frozen by any activities other than the calling activity, return is made with error IU - segment in use.

If there is insuffient free space obtainable in the segment even after squashing all the existing cells together, an attempt to extend the segment is made by a call on EXPAND SEGMENT (see section 8.5.5). If this fails, return is made with error OM - of out memory. If the total size of the cell would be greater than 64K-1 bytes, return is also made with error OM - out of memory.

The cell allocation process also attempts to extend a segment whenever there appears to be insufficient free space for efficient allocation to avoid perpetually having to invoke the garbage collection mechanism. Similarly, if there is a lot of free space in the segment, the size of the segment may be reduced. These changes are of course subject to any size limits imposed on the segment.

8.5.14 REDUCE CELL

Trap Name:	T.KERNEL
Action Value (D0.B):	K.CELLREDUCE

Additional Call Parameters:

D1.W	:	cell tag
D2.W	:	number of bytes to reduce by

Return Parameter:

D0.W	:	cell tag or zero

Error Return:

BP	:	bad parameter

This call reduces the cell by a given amount from the high address end.

If the size of the reduction is less than six bytes, then no action is taken. If the reduced size of the cell would be less than four bytes, the cell is destroyed, and zero returned in D0. Otherwise the original cell tag is returned in D0.

8.5.17 GIVE SEGMENT IDENTIFIER OF CELL

Trap Name:	T.KERNEL
Action Value (DO.B):	K.GIVCSEG

Additional Call Parameter:

D1.W : cell tag

Return Parameter:

DO.L : segment identifier

Error Return:

BP : bad parameter

This call returns the identifier of the segment containing the specified cell.

If the cell tag is zero or otherwise invalid, then return is made with error BP - bad parameter.

8.5.18 RESET CELL ALLOCATOR FOR SEGMENT

Trap Name:	T.KERNEL
Action Value (DO.B):	K.XXXCELL

Additional Call Parameter:

D1.L : segment identifier

Error Returns:

NS	:	invalid segment identifier
IU	:	segment in use (frozen)
BP	:	bad parameter

This Kernel call reinitialises the specified segment as a cell allocator segment. If the specified segment does not exist then return is made with error NS - invalid segment.

The segment is first thawed on behalf of the calling activity. If the segment remains frozen to any other activities then return is made with error IU - segment in use.

If the segment is a cell allocator segment then all the existing cells in the segment are first destroyed.

The segment is then reinitialised as a cell allocator segment.

If the segment remains frozen on behalf of other activities then return is made with error IU - segment in use.

The cells in the segment are all squashed to the high address end, and any free space left after this operation is freed (within the constraints imposed by any minimum segment limit that might be imposed).

Note that it is not normally necessary to obey this call explicitly, as Kernel actions such a sequence periodically on all (thawed) mobile cell allocator segments.

Note also that if this call is used on an immobile cell allocator segment, it will not be possible subsequently to expand the segment back into any space freed by the call.

8.5.21 GET CELL TAG

Trap Name: T.KERNEL
Action Value (DO.B): K.CELLTAG

Additional Call Parameters:

D1.L	:	segment identifier
D2.L	:	offset of cell in segment (in bytes)

Return Parameter:

DO.W	:	tag of specified cell

Error Returns:

NS	:	invalid segment identifier
NF	:	cell not found
BP	:	bad parameter

This call returns the tag of the cell at a given offset from the low address end of a cell allocator segment.

If the segment does not exist, or is not a cell allocator segment, then return is made with error NS - invalid segment identifier. If no cell exists at the given offset, return is made with error NF - cell not found at given address. If a cell is found, the tag of that cell is returned in DO.

The offset of the cell in the segment is passed in D2.L as the number of bytes from the low address end of the segment, and must be in the range 0 to $FFFFFF.

Note that this call involves a search rather than a direct look-up, and hence is not particularly quick.

Error Returns:

none (see below)

This trap call returns the real address of the specified cell. It performs no checking and is designed to be quick.

The relevant segment must be frozen or immobile (and development versions of Kernel may check this).

If this trap is called with a zero or otherwise invalid cell tag value, it will return an address of zero.

The condition codes are unaffected by this trap.

Cells are word aligned.

8.6 Non-Volatile Memory Allocation and Management

The OPD contains 2Kb of CMOS RAM, backed up by battery. This RAM is only accessible from the processor in Supervisor mode [ref 2] and hence must be accessed formally via Kernel.

A certain amount of this RAM is used by Kernel for its own purposes, the remainder is made available to the system and user applications for the permanent storage of important parametric information.

To avoid the necessity of allocating individual addresses once and for all, Kernel effectively maintains a filing system, and provides a formal method of access to the data held in non-volatile memory.

Data in the non-volatile RAM is held as a series of 'entries', which can be from 0 to 255 bytes long. Each entry has an identifier, which is a number in the range 0 to $FFFF.

[It is assumed that ranges of identifiers will be allocated to the various potential users, using manual methods. Such allocation is beyond the scope of this document.]

Kernel maintains a checksum of the information stored in the non-volatile memory. This checksum is validated as part of the establishment checks, and the initial activity informed if there has been any corruption (see section 20.3).

Note that the overheads on accessing non-volatile memory via Kernel imply that access is infrequent.

specified in the call, the data in the non-volatile memory is updated from the caller's buffer. If the specified entry exists, but with a different length of that specified in the call, return is made with error OR - out of range (incorrect length) - and the non-volatile data left unaltered.

If the specified entry does not exist, and there is room for it, the entry is created, and the data copied from the caller's buffer. If there is insufficient room to create the new entry, return is made with error OM - out of memory.

For zero length entries, the buffer address is ignored.

Note that all entries have 3 bytes of red tape, and a request to create a new zero length entry can fail because there is insufficient space.

8.7.3 DELETE NON-VOLATILE MEMORY ENTRY

Trap Name:	T.KERNEL
Action Value (D0.B):	K.XNVM

Additional Call Parameters:

D1.W	:	entry identifier	(0 to $FFFF)
D2.W	:	entry length	(0 to 255)

Error Responses:

NF	:	the specified entry does not exist
OR	:	incorrect length specified
BP	:	bad parameter

If the specified entry exists, and has the same length as specified in the call, it is deleted from non-volatile memory.

If the entry does not exist, error NF - entry not found - is returned. If the entry exists but has a different length, error OR - out of range (incorrect length) - is returned.

routine). Physically it comprises some internal workspace and parameters for the device routine, and possibly some internal buffering. The space used for this purpose is provided by Kernel, and is known as the Channel Control Area.

An activity may have more than one channel open at any time. In some circumstances, more than one channel (belonging to the same or different activities) can be open to a particular device.

An activity requests the setting up of channel by making an 'open' call on Kernel, and indicates that it has finished by making a 'close' call. A channel may be opened on behalf of another activity if desired.

(Note than once a channel has been opened, further system considerations may prevent the channel being used at certain times - this applies particularly to accessing the screen and keyboard.)

9.2.1 CHANNEL CONTROL AREAS

When an OPEN CHANNEL call is made to allocate a device or file and set up a channel, a Channel Control Area is created in a cell allocator segment belonging to Kernel. This area of RAM is used by the relevant device routine for private workspace, buffering and the like. If it is not possible to create the space, then error OM - out of memory - is returned.

The length of a given channel control area depends primarily on the type of device, and in some cases on the amount of internal buffering required. Except where a large amount of channel buffering is required, the creation of a channel control area should not fail unless memory is in extremely short supply.

9.2.2 CHANNEL IDENTIFIERS

Each currently open channel has a unique identifier, which is used whenever a reference to that channel is required. The channel identifier is returned from an OPEN CHANNEL call and remains a valid identifier until a CLOSE CHANNEL call is made.

Physically a channel identifier consists of a long word with a unique positive and non-zero value in the top word, and a channel number in the bottom word. The unique values are allocated sequentially by Kernel. The channel number is an index into Kernel's tables, and may be reused if the channel is closed.

A channel identifier may only be used by the activity associated with the channel, or the activity that opened the channel.

9.3 Input/Output Operations

The normal input and output operations available on a channel are as follows:-

9.5 Channel Directionality

Most devices or files are uni-directional (i.e. they support input only or output only). In such cases, if an invalid I/O operation is attempted, then error NI - operation not implemented - is returned and there is no other adverse effect on the device or channel.

When a channel or device is capable of supporting both input and output, but in 'half duplex' mode, then the current direction of the channel is explicitly or implicitly turned around by an I/O call.

Channels that support simultaneous input and output (i.e. true 'full duplex' mode) provide sufficient internal buffering to avoid any problems caused by the restriction that only one operation may be in progress on a channel at once. In some circumstances, it is possible to open both an input and an output channel to the same device.

9.6 Channel Error Handling

To a large extent, the channel error handling depends on the device or file to which it is connected. In some cases, very little checking or error reporting is possible; in others errors can arise out of the data passed to the channel. Moreover, with the asynchronous nature of some of the devices, problems can occur when no I/O operation is actively in progress on the device.

To attempt to alleviate these problems in some way, Kernel distinguishes between two major types of channel error:

a) fatal or unrecoverable errors on the device or channel, such that further I/O cannot proceed without some kind of intervention; these are known elsewhere in this document as device errors

b) non-fatal or recoverable errors on the device or channel of the kind that allow an I/O operation to proceed (as far as the Kernel device routine is concerned); these are known elsewhere in this document as device or channel warnings

If a device error occurs during an I/O operation, then the operation is immediately terminated. A GIVE CHANNEL STATUS or CANCEL I/O TRANSFER call can be used to determine the type of error; this call also clears the error notification from the channel. If an I/O operation is attempted on a channel with an unreported device error (because a problem has occurred subsequent to the last GIVE CHANNEL STATUS or CANCEL I/O TRANSFER call) then the I/O is not initiated, but fails immediately with an error return DE - unreported device error.

[In practice this means that if termination events are used, a

Register A1 is used to point to further information required by the OPEN CHANNEL call (such as a file name). In some circumstances A1 may be set to zero, whereupon a default set of further information is used. Not all devices require any further information, in which case the contents of A1 are ignored; details of the extra information required and any defaults will be found in the appropriate device routine specifications.

The open call is processed as follows:-

.1) if it is possible to determine immediately that access to the specified device or file is not possible or permitted, then the call makes an immediate error return

.2) if it is possible to determine immediately that access to the specified device or file is possible and permitted, then a Channel Control Area is created, and the call makes an immediate successful return with a new channel identifier. If there is no room for a new CCA, then error OM - out of memory - is returned, and the device or file remains unallocated

.3) if the success of the open is impossible to determine immediately, then a Channel Control Area is created as above, and the call returns with error NC - open not complete. The Open operation is queued in the channel control area in the manner of a normal queued I/O operation: a valid channel identifier is returned

.4) following a return with error NC, when the open operation does finally complete, an event will be generated if specified at the time of the call, and the success or failure of the open may be determined from a GIVE CHANNEL STATUS call

.5) if the delayed open proved to have failed, the channel identifier returned will only be valid for GIVE CHANNEL STATUS calls and CLOSE CHANNEL calls, and a CLOSE CHANNEL call should be made to free the channel and delete the Channel Control Area

Note that delayed OPEN CHANNEL calls are the exception, and individual device routine specifications will give details of the circumstances in which they might occur. Note also that not all the errors listed above will necessarily be generated by any given device routine.

9.7.2 CLOSE CHANNEL

Trap name:	T.KERNEL
Action Value (D0.B):	K.CLOSE

NI : operation not implemented on this channel
BP : bad parameter

This call gets (reads) a single byte from the specified channel directly into the caller's D0, and makes an immediate return.

It may only be made by the activity associated with the channel; any other activity making this call will cause a return with error NO - invalid channel identifier.

If a byte is not immediately available from the channel, then return is made with error NB - no byte immediately available. In this latter case, then if or when a byte subsequently becomes available, the event specified at the time of the call will be signalled to the activity, and the next GET BYTE IMMEDIATE operation will return the byte. The event will also be signalled if a device error occurs or the channel is unexpectedly closed for any reason. If no event is required, then the value -1 may be passed to the call. Note that only one occurrence of the event will be signalled as a result of any GET BYTE IMMEDIATE call.

If, following a return with error NB, it is decided that the event is no longer required, the effect can be cancelled by a CANCEL I/O TRANSFER call or any queued I/O operation.

If the channel already has a queued operation in progress, return is made with error IU - channel in use - and no other effect on the channel's behaviour is made.

If the channel has had a device error occurring since the last GIVE CHANNEL STATUS or CANCEL I/O TRANSFER call was made, then return is made with error DE - unreported device error. The next GIVE CHANNEL STATUS or CANCEL I/O TRANSFER call will return further information about the failure, and clear the error from the channel.

Note that efficiency considerations may preclude the reading of large volumes of data via this call.

9.7.4 PUT BYTE IMMEDIATE

Trap name:	T.KERNEL
Action Value (D0.B):	K.PUTBYTE

Additional Call Parameters:

D1.L : channel identifier
D2.B : byte to be written
D3.W : event number or -1 (see section 6.4)

Error Returns:

NO : invalid channel identifier or channel not open

Error Returns:

NO	:	invalid channel identifier or channel not open
IU	:	channel in use (queued operation pending)
NI	:	operation not implemented on this channel
DE	:	unreported device error
BP	:	bad parameter

This call queues an I/O transfer to get (read) the specified number of bytes into the caller's buffer.

It can only be made by the activity associated with the channel; any other activity making this call will cause a return with error NO - invalid channel identifier.

The call returns immediately, and if a success code is returned the transfer will proceed autonomously, terminating when the specified number of bytes has been read, or a device error occurs, or the transfer is cancelled for any reason.

If the channel already has a queued operation in progress, return is made with error IU - channel in use - and no other effect on the channel's behaviour is made.

If the channel has had a device error occurring since the last GIVE CHANNEL STATUS or CANCEL I/O TRANSFER call was made, then return is made with error DE - unreported device error. The next GIVE CHANNEL STATUS or CANCEL I/O TRANSFER call will return further information about the failure, and clear the error from the channel.

The buffer length in D2.W must be in the range +1 to +16383.

Following a successful return from this call, then if or when the transfer terminates or is unexpectedly terminated by another activity or Kernel, the event specified at the time of the call is signalled to the activity. If the event number passed to the call was -1, no event will be signalled.

A GIVE CHANNEL STATUS call may be made at any time to determine the current state of the channel, and (on termination) whether the transfer suceeded.

9.7.6 QUEUED GET STRING WITH TERMINATOR

Trap name:	T.KERNEL
Action Value (D0.B):	K.QGETTERM

Additional Call Parameters:

D1.L	:	channel identifier
D2.W	:	buffer/string length (in bytes)
D3.W	:	event number or -1 (see section 6.4)

the channel's behaviour is made.

If the channel has had a device error occurring since the last GIVE CHANNEL STATUS or CANCEL I/O TRANSFER call was made, then return is made with error DE - unreported device error. The next GIVE CHANNEL STATUS or CANCEL I/O TRANSFER call will return further information about the failure, and clear the error from the channel.

The buffer length in D2.W must be in the range +1 to +16383.

Following a succesful return from this call, then if or when the transfer terminates or is unexpectedly terminated by another activity or Kernel, the event specified at the time of the call is signalled to the activity. If the event number passed to the call was -1, no event will be signalled.

A GIVE CHANNEL STATUS call may be made at any time to determine the current state of the channel, and (on termination) whether the transfer succeeded.

9.7.8 GIVE CHANNEL STATUS

Trap name:	T.KERNEL
Action Value (D0.B):	K.GIVCHAN

Additional Call Parameter:

D1.L	:	channel identifier

Return Parameters:

D0.W	:	logical device number
D3.L	:	device status information
D2.W	:	unexpired buffer count (in bytes)
A1	:	current buffer pointer

Error Returns:

NO	:	invalid channel identifier or channel not open
BP	:	bad parameter

This call returns information concerning the current state of the channel, as described below, and may be made at any time following the opening of a channel. It may only be made by the activity associated with the channel; any other activity making this call will cause a return with error NO - invalid channel identifier.

The information returned is as follows :-

a) The logical device number returned in D0 is that number passed to the OPEN CHANNEL call.

portion of the buffer.

If the channel/device status returned by this call reports a device error, then the error indication is cleared from the channel and another GIVE CHANNEL STATUS call will return a zero status (unless a further device error occurs in the interim). Any channel or device warnings reported by this call are cleared from the channel even if an I/O operation is in progress. Channel information flags are never cleared by this call.

9.7.9 CANCEL I/O TRANSFER

Trap name:	T.KERNEL
Action Value (DO.B):	K.XIO

Additional Call Parameter:

D1.L : channel identifier

Return Parameters:

DO.W	:	logical device number
D3.L	:	device status information
D2.W	:	unexpired buffer count (in bytes)
A1	:	current buffer pointer

Error Returns:

NO	:	invalid channel identifier or channel not open
BP	:	bad parameter

This call cancels any outstanding I/O transfer on the channel and returns the channel status in the manner of a GIVE CHANNEL STATUS call.

It can only be made by the activity associated with the channel; any other activity making this call will cause a return with error NO - invalid channel identifier.

If there is a queued transfer in progress, the device routine is informed, and the transfer terminated immediately. The channel status is set to zero (no I/O in progress).

If no queued transfer is in progress, but an event has been requested as a result of a PUT or GET BYTE IMMEDIATE call the outstanding event request is cancelled, but no other action taken.

Detailed information on any effects of this call on the outstanding I/O will be found in the relevant device routine specifications.

On return from this call, the registers are set as for a GIVE CHANNEL STATUS call (see section 9.7.8) the channel/device status

10. SCREEN DEVICE ROUTINE

10.1 Basic Device Characteristics

10.1.1 HARDWARE OUTLINE

The OPD screen is a bit-mapped device capable of operating in two modes, high resolution mode and low resolution mode. In high resolution mode, 512 x 256 pixels are displayed, where each pixel can be one of 4 different colours. In low resolution mode, 256 x 256 pixels are displayed, where each pixel can be one of 8 different colours, and further, has the ability to flash. The colours include black and white, and are represented on monochrome displays as different intensities. (See [ref 2].)

10.1.2 LOGICAL PIXEL ADDRESSING

All pixel addresses (in either mode) are defined as X and Y coordinates in a 512 x 256 logical display map, where the top left corner of the screen is pixel position (0,0).

This means, for example, that the middle of the screen is pixel position (256,128) in both hardware display modes.

Pixel addressing is normally of use only in the graphics Kernel calls, and the co-ordinates are absolute (i.e. relative to the real screen origin).

10.1.3 WINDOWS

The majority of operations carried out by the screen device routine are character oriented, and performed on the currently defined window, where a window is a rectangular subset of the total screen area available to a particular channel (see below).

Windows are defined in terms of their position and size by specifying the character positions (i.e. row and column) of the top left corner and the bottom right corner, relative to the screen area available to the channel. The actual screen area available to a channel can be altered if desired, allowing characters to be written at any pixel position on the screen.

Note that if a new window is defined to overlap an existing window, the contents of the latter's overlapping area are not saved, and will be overwritten if the new window is written to.

When a channel to the screen is created, a default window is set up; this may be changed, but there remains a one to one correspondence between windows and screen channels.

10.1.4 CHANNEL DISPLAY MODES

The hardware provides for two resolution modes, as described above. The screen driver device routine extends this to some

left and top (see Appendix F for further details). [This font size enables the display (in theory) of 25 lines of 42 characters, (low-res mode) or 25 lines of 85 characters (high-res mode).]

In 64 column mode, the font size is 6 x 9 character shapes in a 7 x 10 frame (see Appendix F for further details).

When the vertical display mode is configured to 8 pixel mode, only the middle 8 vertical band of pixels in the font are used, resulting in some truncation at the top and bottom of the character.

On any given channel I/O transfer two fonts are available - the primary font and the secondary font. The primary font can have up to 128 character shapes (accessed by codes in the range $20 to $9F); the secondary font can have up to 96 character shapes (accessed by codes in the range $A0 to $FF). The default primary font contains the normal 96 ISO graphic characters (see Appendix E). The default secondary font contains some additional character shapes used by the OPD software (see Appendix F).

Either of the fonts used by a channel may be set to a user-supplied font - details of the font format are given in Appendix F.

10.1.6 CHARACTER SIZE ATTRIBUTES

As well as defining the channel display modes and the fonts used to display graphic characters, within any given font it is possible to adjust the size of the character actually generated. The various character size attributes that can be specified are as follows:-

a) Single Height - these characters are the same height as the image in the font, and occupy 10 or 8 pixel lines in the downwards vertical direction (including the guard band normally included at the top of each character).

b) Double Height - these characters are twice the height of the character image in the font, and occupy 20 or 16 pixel lines in the downwards vertical direction. Each line in the font is written to two consecutive pixel lines.

c) Single Width - in 80 and 64 column mode these characters occupy 6 or 7 logical pixel columns in the horizontal direction (including the guard band normally included at the left of each character). In 40 column mode and Compatible 40 column mode single width characters are not supported, (as all characters in this mode are effectively double width).

d) Double Width - in 80 and 64 column mode these characters are twice the width of the equivalent single width characters. Double width characters occupy 12 or 14 logical pixel

When Cursor Wrap leaves the cursor on the same line, it is known as a 'Toroidal' wrap; when the cursor moves to a different line it is known as a 'Progressive' wrap.

Any combination of these cursor actions at window boundaries may be selected as desired.
If a particular boundary is reached, and no boundary handling is specified for it, then the cursor position is not moved and any further graphic characters in the transfer will be written at that cursor position (unless the cursor is deliberately repositioned by a subsequent control character).

Note that in double height or double width mode the boundary handling works as one would expect, even with windows of an odd number of rows and columns, scrolling or wrapping when it is not possible to fit another character on in the relevant direction. (Note that if double height or double width is set, the cursor can never be positioned in the last row/column.)

10.1.8 COLOURS

The system provides for 8 colours, encoded as follows:

	Low-Res	High-Res
0	black	black
1	blue	black
2	red	red
3	magenta	red
4	green	green
5	cyan	green
6	yellow	white
7	white	white

On monochrome displays the colours are represented as shades of grey, with the intensity increasing in the above order.

If the compatible 40 column channel display mode is used and consistent results are required, only the colours numbered 0 (black), 2 (red), 4 (green) and 7 (white) should be used.

10.1.9 PLOTTING MODES

Normally characters, lines and points are 'plotted' on the screen by using the current ink colour (foreground) on the current paper colour (background). Two other modes are possible, 'transparent' and 'exclusive-or'.

In 'transparent plot mode', only the foreground is affected, and the background remains unaltered.

In 'exclusive-or plot mode', the foreground colour is written

Default attributes are set up whenever a channel to the screen is opened, and are subsequently used whenever that window is selected. If the attributes of a channel are changed, then that change affects all subsequent operations on the channel, but not the existing contents of the screen.

10.1.13 SCREEN BLANKING

If there has been 'no activity' on the OPD for a period of 10 minutes then the screen device routine will blank the entire screen to avoid phosphor burn. The display will be restored when any of the occurrences below are detected by Kernel. This screen blanking is transparent to any activity using the screen.

Occurrences which cause the screen to be restored are:

a) any change of state on the telephone lines

b) any change of state on the keyboard (including the depression of any shift key alone)

c) any I/O transfers to an application screen channel

d) a particular kind of I/O transfer to the noticeboard channel

10.2 Screen Areas and Window Origins

In software terms the screen is primarily divided into two logical areas, one large and one small. The area comprising the top 240 pixel lines is called the Application Screen Area, (or more commonly just the screen). It constitutes the majority of the screen, and can be freely used by those activities entitled to access the screen at any particular time, usually for displaying 24 lines of single height ten pixel mode characters.

The remaining 16 pixel lines at the bottom of the screen form an area called the Noticeboard, and are used to display status information by the OPD system software, using two lines of single height eight pixel mode characters.

These two areas are presented as different logical devices. All the facilities for writing characters in different modes etc. apply to both the application screen and the noticeboard.

10.2.1 CHANNEL SCREEN AREAS

Within the Application Screen Area or Noticeboard Area, each channel defines a rectangular subset within which its operations are constrained. This area is defined in terms of the pixel coordinates of the top left and bottom right corners. This channel screen area is defined implicitly or explicitly at the time the channel is opened, but may subsequently be changed if

alternative formal access methods. The effect of writing to the noticeboard area is undefined (as far as Kernel is concerned). The effect of writing to the high address screen map area of RAM is guaranteed to be catastrophic.

10.6 Transfer Timing and Timeouts

Although there are no actual hardware limitations on accessing the screen (being completely governed by processor and memory speed), it should be noted that updating the screen can take a noticeable time, and there may be system considerations that further affect the issue (like microdrive activity and special key events).

Furthermore, the CPU requirements of queued serial I/O calls upon screen channels are scheduled in a manner similar to activities, and hence may take a significant time to complete. Any timeouts must make due allowance for this.

10.7 Serial I/O Operations

Via the standard serial I/O interfaces, it is possible to write strings of text and control information. The strings may be a freely chosen combination of the following:-

a) primary graphic characters
b) secondary graphic characters
c) control code sequences

and these are now described in more detail.

10.7.1 PRIMARY GRAPHIC CHARACTERS

This category comprises displayable characters with codes in the range $20 to $9F (although the total range is not currently used). When such a character is written to a screen channel, the relevant entry from the primary font is exploded into the screen store, with the current window attributes being added/ incorporated.

The cursor is then moved right by one character width. If this would take the character off the edge of the window, then the specified form of boundary handling is performed. If neither form of Auto Cursor-Wrap is specified for the right edge of the window, the cursor will remain in the same position, and the next graphic character will overwrite the previous one.

The character shapes in the default primary font are shown in Appendix E and Appendix F.

10.7.2 SECONDARY GRAPHIC CHARACTERS

This category comprises displayable characters with codes in the range $AO to $FF, and they are handled in a manner analogous to

10.7.3.7 Newline (NL)

This performs the same actions as Carriage Return followed by Line Feed.

10.7.3.8 Home Cursor (HOME)

The cursor is moved to the top left character position of the current window.

10.7.3.9 Clear Window/Form Feed (CLRW or FF)

Under normal circumstances, the current window is cleared (set to background/paper colour) and the cursor homed. If the channel screen area defined by an OPEN CHANNEL or SET CHANNEL DISPLAY MODES call is not an exact number of characters wide or deep, and no subsequent window redefinition has occurred, then the area cleared will be the the channel screen area, which may be slightly larger than the current window (or in one unusual case, slightly less). Further details will be found in sections 10.1.2, 10.7.3.28 and 10.10.5.

[This obscure behaviour allows the quick colour filling of any sized rectangular areas anywhere on the screen.]

10.7.3.10 Delete Character (DELC)

This control code normally deletes (i.e. clears to blank) the character to the left of the cursor. The effect is achieved by simulating the sequence Backspace/Space/Backspace. The space is not underlined. Note that with certain boundary condition handling requirements this control code will have a somewhat different effect when called with the cursor at the left hand edge of the window.

10.7.3.11 Invert Ink and Paper Colours (INV)

The current ink/foreground colour is swapped with the current paper/background colour.

10.7.3.12 Display Cursor (CURD)
10.7.3.13 Inhibit Cursor (CURI)

These control codes switch the display of the current channel's cursor on and off respectively. The cursor position is not affected. Note that when strings of text are written to the channel, the cursor display is removed from the screen and subsequently displayed (or not) at the end. The cursor is also removed/inhibited by CLOSE and SET CHANNEL DISPLAY MODE calls.

hand boundary irrespective of any boundary handling attribute.

Note that the final position depends on the screen mode and the current character width attribute.

10.7.3.17 Double Height (DBLH/n)

This control code is used to switch between Single and Double Height characters. If the single parameter byte is 1, then subsequent characters will be double height; if it is zero, subsequent characters will be single height. If the current cursor position is on the bottom (single height) row of the window, then an attempt to switch the window into double height will terminate the transfer with device error OR - out of range.

If the cursor is displayed, its height will be altered to match the new height attribute; the top left corner of the cursor remains unmoved.

10.7.3.18 Double Width (DBLW/n)

This control code is used to switch between Single Width and Double Width characters in 80 or 64 column channel display modes. If the single parameter byte is 1 then subsequent characters will be double width; if it is zero, subsequent characters will be single width. If the current cursor position is in the (single width) right hand column of the window, then an attempt to switch the window into double width will terminate the transfer with device error OR - out of range. In 40 column and compatible 40 column channel display modes, this control code sequence is ignored (as such windows are always effectively in double width).

If the cursor is displayed, its width will be altered to match the new width attribute; the top left corner of the cursor remains unmoved.

10.7.3.19 Set Character Plot Mode (CPLOT/n)

This control code is followed by a single parameter byte which defines the required character plot mode as follows:

0	normal (ink on paper colour)
1	transparent (ink on transparent background)
2	exclusive-or

For further details see section 10.1.7. Note that flashing gives unusual results except in normal plot mode.

- **Clear to End of Window (ESC/CLREW)**

  Clears from the cursor position to the end of the window. The cursor position is unaltered.

- **Save Cursor Position (ESC/SVCUR)**

  Saves the current cursor position. The previously saved position (if any) will be lost.

- **Restore Cursor Position (ESC/RSCUR)**

  Restores the cursor position from the previously saved position.

- **Partial Scroll 1 (ESC/PSCR1)**

  Scrolls the window from the top of the cursor to the top of the window up one character height. The cursor position is unaltered.

- **Partial Scroll 2 (ESC/PSCR2)**

  Scrolls the window from the top of the cursor to the bottom of the window down one character height. The cursor position is unaltered.

- **Partial Scroll 3 (ESC/PSCR3)**

  Scrolls the window from the top of the window to the top of the cursor down one character height. The cursor position is unaltered.

- **Partial Scroll 4 (ESC/PSCR4)**

  Scrolls the window from the bottom of the window to the top of the cursor up one character height. The cursor position is unaltered.

- **Prestel Flash Left (ESC/PFL)**
- **Prestel Flash Right (ESC/PFR)**
- **Special Prestel Flash Left (ESC/XPFL)**
- **Special Prestel Flash Right (ESC/XPFR)**

  Set the flash bits in the left-most pixels (PFL, XPFL) or the right-most pixels (PFR, XPFR) of the character at the current cursor position (see section 10.1.10 and [ref 2]). For the PFR and PFL sequences, the relevant pixels are also set to the current Paper colour. These escape sequences should normally be followed by a control code that moves the cursor. These escape sequences are ignored except when the channel is in (genuine) 40 column mode.

The cursor (if displayed) is removed from the window. The window is then redefined as specified, and the cursor position and saved cursor positions are homed. The character size attributes are set to single height and single width (80 and 64 column modes) or double width (40 column modes). No other visible effects occur. Note that the row and column number of a given screen position varies depending on the current channel display mode and channel screen area.

Note also that the character positions of the right and bottom edges as specified by this control code are **outside** the window so that e.g. the default application screen window would be defined in 80 column, 10 pixel channel display mode by the string

```
WIND/0/0/80/24
```

Finally, the bottom and right hand edges of the current channel screen area are adjusted to coincide with the new window area.

10.7.3.29 Paint (PAINT/l/t/r/b/io/po/in/pn)

This control sequence is followed by 8 parameter bytes, and is used to change colours within a rectangular area.

The first four parameter bytes define a 'temporary' window like area, relative to the current window, and in the manner described for Redefine Window, except that the character positions defined by the first four bytes take account of the character size attributes as well as the channel display modes.

The next two parameters specify the old colours (old ink and old paper).

The last two parameters specify the new colours (new ink and new paper).

Within the temporary window, any pixel that is of the old ink colour is changed to the new ink colour, and any pixel that is of the old paper colour is changed to the new paper colour.

This facility is restricted when the channel is operating in low resolution hardware display mode, in that the colours are handled as if numbered as for high resolution mode, and hence e.g. both black and blue would be treated the same.

If the temporary window defined exceeds any dimension of the current window, the transfer is terminated with device error OR - out of range.

activity associated with the Noticeboard channel must be a system activity.

10.9.2 CLOSE CHANNEL

Call Parameters and Error Returns:

see section 9.7.2

A CLOSE CHANNEL call to an application screen channel will remove the cursor from the current window if it is displayed, but otherwise there is no effect on the displayed screen image.

This call is not implemented for the Noticeboard channel and will return error NI - operation not implemented. The effect of any subsequent I/O call on the channel is undefined.

10.9.3 GET BYTE IMMEDIATE

This call always returns error NI - operation not implemented.

10.9.4 PUT BYTE IMMEDIATE

Call Parameters and Error Returns:

see section 9.7.4

This call always makes a successful or device error return. It never returns error NB - no byte put. For a description of possible device errors see GIVE CHANNEL STATUS below.

The device routine first checks that the hardware display mode is correct for the current channel display mode. If it is, the transfer proceeds. If it is not, and the horizontal channel display mode is 'compatible 40 column', the window definition block is reconfigured before proceeding with the transfer. Otherwise the transfer will fail with device error SR - screen resolution incorrect.

10.9.5 QUEUED GET STRING
10.9.6 QUEUED GET STRING WITH TERMINATOR

These calls always return error NI - operation not implemented.

10.9.7 QUEUED PUT STRING

Call Parameters and Error Returns:

see section 9.7.7

The device routine first checks that the hardware display mode is correct for the current channel display mode. If it is, the transfer proceeds. If it is not, and the horizontal channel display mode is 'compatible 40 column', the window definition

and D2.W) to restart the transfer if desired.

10.9.10 CHANGE CHANNEL OWNERSHIP

Call Parameters and Error Returns:

see section 9.7.10

10.10 Additional Screen Driver Kernel Calls

10.10.1 FAST PUT LINE IMMEDIATE

Trap Name:	T.KERNEL
Action Value (DO.B):	K.FASTLN

Additional Call·Parameters:

D1.L	:	channel identifier
D2.W	:	buffer/string length (in bytes)
A1	:	address of buffer

Error Returns:

NO	:	invalid channel identifier or channel not open
IU	:	channel in use (queued operation pending)
DE	:	device error
BP	:	bad parameter

This call writes a single line of text in 80 column mode, but makes certain assumptions about the data string being written to make it run at approximately twice the speed. The call behaves in a manner analagous to PUT BYTE IMMEDIATE and makes an immediate return having written the line.

The assumptions made by this call are as follows:

a) there are no control characters in the buffer
b) the cursor is already positioned at the correct place
c) the cursor display is inhibited
d) the character size attributes are single width and single height
e) the specified number of characters will not cause the line to go off the right hand edge of the window

The cursor position is not updated by this call. If any of the assumptions made are false or the horizontal channel display mode is other than 80 column, the result is undefined and potentially catastrophic.

10.10.2 CHANGE CHARACTER FONTS

Trap Name:	T.KERNEL
Action Value (DO.B):	K.FONT

Please purchase Image2PDF on http://www.verypdf.com/ to remove this message.

This Kernel call plots a single pixel point of specified colour. The call is **not** window based, and the coordinates specified are absolute pixel address (see section 10.1.2).

The call behaves in a manner analogous to PUT BYTE IMMEDIATE, and makes an immediate return having plotted the point.

A1 points to a 6 byte word aligned buffer that has the following contents:

x	x	y	y	p	c

x : x coordinate

y : y coordinate

p : plot mode (see section 10.1.9)
0 plot colour on pixel position
1 'or' colour, with pixel position
2 'exclusive-or' colour into pixel position

c : colour of pixel

The x coordinate is a logical pixel number (taken modulo 512), and in low-resolution display mode it is rounded down to the nearest even number. The y coordinate is a logical pixel number (taken modulo 240 for Application Screen channels, and modulo 256 for the Noticeboard channel).

10.10.4 DRAW LINE IMMEDIATE

Trap Name: T.KERNEL
Action Value (D0.B): K.DRAW

Additional Call Parameters:

D1.L : channel identifier
A1 : buffer address

Error Returns

NO : invalid channel identifier or channel not open
IU : channel in use (queued operation pending)
DE : unreported device error
BP : bad parameter

This Kernel call draws a straight line from one pixel position to another. The call is **not** window based, and the coordinates are absolute pixel addresses (see section 10.1.2).

The call behaves in a manner analogous to PUT BYTE IMMEDIATE, and makes an immediate return having drawn the line.

bits 0 - 1 horizontal mode

0	:	80 column (high resolution)
1	:	64 column (high resolution)
2	:	40 column (low resolution)
3	:	compatible 40 column (either resolution)

bit 7 vertical mode

0	:	10 pixel high characters
1	:	8 pixel high characters

Bit 6 of D2.B is used as a flag to indicate if the channel screen area is to be defined (bit 6 = 1) or whether the default appropriate to the new channel display modes is used (bit 6 = 0). The defaults are described in section 10.1.4.

If bit 6 is set, then A1 contains the address of an 8 byte word-aligned area as follows:

```
|l|l|t|t|r|r|b|b|
```

l	:	pixel position of left edge	(0 to 511)
t	:	pixel position of top edge	(0 to 239)
r	:	pixel position of right edge	(1 to 512)
b	:	pixel position of bottom edge	(1 to 240)

This area defines the required channel screen area in terms of the absolute pixel positions of the edges (where the right and bottom edge pixel positions are outside the area in a manner similar to window boundaries). If bit 6 is zero or the channel is a Noticeboard channel, then the contents of A1 are ignored.

The channel screen area and channel display modes are set up, and the window boundaries recalculated to cover the channel screen area as closely as possible. In particular the top left hand corner of the window coincides with the top left corner of the implicit or explicit channel screen area. (See section 10.1.2.)

Note that this call effectively performs a Close operation followed by an Open with the required display modes. Consequently, all the window and character attributes, fonts etc. are reset to the relevant default values as described in Appendix F.

Note that it is possible to define an application screen area less than one character high or wide. If such an area is used, then any attempt to write graphic characters will act as though the window defined within the screen area is in fact at least one character high and/or wide.

No check is made that the screen hardware is in the correct mode

```
bit 1  :  0 => display operating
          1 => display blanked (because no activity)
bit 3  :  0 => 512 pixel mode (high resolution)
          1 => 256 pixel mode (low resolution)
```

10.10.8 SET HARDWARE DISPLAY MODE

Trap Name: T.KERNEL
Action Value (D0.B): K.SETHDISP

Additional Call Parameter:

D1.B : new display hardware mode

Error returns:

none

This call changes the hardware display mode to that specified. The new mode is encoded in D1.B as follows :-

bit 3 zero : 512 pixel mode (high resolution)
bit 3 non-zero : 256 pixel mode (low resolution)

The action of this call is as follows:

a) the hardware blanking feature is enabled

b) the whole screen store area is zeroed (i.e. set to 'black')

c) the hardware mode is set to the required value and the screen unblanked

d) the global event 'screen mode changed' is signalled

These actions take place even if the hardware is already in the required mode. This call should obviously be used with care and discretion.

subsequent key combinations.

11.3.2 TELEPHONE

This category comprises the telephone control key combinations, namely:

RECALL
SPEAKER
REDIAL
HOLD & SHUTTLE
SELECT
END
DIAL
HOLD
TIME
AUTO

These key combinations are presented as part of the telephone device(s) and do not appear on keyboard channel input.

11.3.3 KEYPAD NUMERIC

This category comprises the keys on the numeric keypad, without any Shift/Control/ALT keys in combination, i.e.

0 to 9
*
#

These key combinations are sometimes presented as part of the keyboard, and sometimes as part of the telephone device(s) in a manner transparent to the keyboard channels.

11.3.4 ORDINARY NUMERIC

This category comprises the numeric codes generated from the unshifted punctuation row at the top of the keyboard. The codes from this category are the same as for the keypad numerics, but the routing of this category is different.

11.3.5 OTHER GRAPHIC

This category comprises all the other codes representing characters from the graphic set.

11.3.6 CAPS LOCK

This category comprises the CAPS LOCK key combination only and the key code(s) are always passed to the system keyboard without changing any subsequent routing of key combinations.

the OPD high level scheduler/Director software.

Secondly, the system may be in 'shortcode dial state' or 'normal dial state'. Entry to shortcode dial state or dial state is controlled by the telephone handling software, and exit from either dial state is likewise controlled.

The routing of the various key codes in the various system states is shown in figure 11.1 at the end of this chapter.

[Note that this mechanism assumes that the telephony handling software responds quickly enough to perform any switching before the next character needs routing. Whether this is satisfactory or achievable remains to be determined.]

Note also that input from the normal keyboard is sent to the current normal keyboard. The decision over which of several normal keyboard channels is current is made by the owners of the channels, in cooperation with each other. If there is no normal keyboard currently selected, then the characters are routed to the system keyboard channel.

At initialisation/power up reset time, the system is in System Key processing mode, and neither dial state is current.

11.6 System Control Keys and Screen Handling

As already described, the system control key combinations perform certain routing operations. A further effect of system control keys is that on entry to system key processing mode, any queued transfer outstanding on an application screen channel will be cancelled.

Further, while the system is in system key processing mode, any transfers to the application screen started by untrusted activities will cause the program counter to be stepped back, and another activity scheduled.

11.7 The Caps Lock Key Combination

This key combination behaves in the conventional manner, affecting the codes generated by single key depression of the alphabetic keys. In order that the higher-level OPD software can maintain a Caps Lock flag in the noticeboard, Kernel alternately generates two different key codes - Caps On and Caps Off; these are routed to the system keyboard channel.

Following an initialisation sequence, the Caps Lock key is considered to be in the unshifted state.

11.8 Keyboard Buffering

All keyboard channels provide a measure of internal buffering of input characters. The amount needed depends on the amount of

follows:

```
|b|b|s|s|

b : channel buffer size (1 - 255)
s : selection criteria (see below)
```

The b field specifies the size of the internal channel buffer in bytes, and must be in the range 1 to 255.

If the selection criteria word (s field) is non-zero, a SELECT NORMAL KEYBOARD CHANNEL call will be simulated immediately after the succesful open (see below). The value of this field is ignored when opening a channel to a system keyboard.

11.11.2 CLOSE CHANNEL

Call Parameters and Error Returns:

see section 9.7.2

This call closes a normal keyboard channel in the usual manner, as described in section 9.7.2.

This call is not supported for the system keyboard, and will always return error NI - operation not implemented. The effect of any subsequent I/O call on the channel is undefined.

11.11.3 GET BYTE IMMEDIATE

Call Parameters and Error Returns:

see section 9.7.3

11.11.4 PUT BYTE IMMEDIATE

This call always returns error NI - operation not implemented.

11.11.5 QUEUED GET STRING

Call Parameters and Error Returns:

see section 9.7.5

11.11.6 QUEUED GET STRING WITH TERMINATOR

Call Parameters and Error Returns:

see section 9.7.6

Director imposes certain rules concerning the use of this Kernel call.

11.12.2 CHANGE KEYBOARD PROCESSING MODE

Trap Name:	T.KERNEL
Action Value (D0.B):	K.SETKBM

Additional Call Parameter:

D1.W : change required (see below)

Error Returns:

BP : bad parameter

This call changes the current processing mode of the keyboard device routine. The change required is encoded as follows:

```
D1.W  =  0 : enter system key mode
         1 : exit system key mode
         2 : enter shortcode dial mode
         3 : enter dial mode
         4 : exit either dial mode
```

This call can only be made by System activities.

Note that on exit from system key processing mode, there is no current normal keyboard established until a SELECT NORMAL KEYBOARD CHANNEL is performed (or an OPEN CHANNEL call with the Select option), and any characters arriving in this period will be routed to the system keyboard.

11.12.3 SET REPETITION AND DELAY TIMES

Trap Name:	T.KERNEL
Action Value (D0.B):	K.KEYRPT

Additional Call Parameters:

D1.W : new delay value
D2.W : new repeat value

Error Return:

BP : bad parameter

This call enables the key repetition parameters to be altered. The initial delay and subsequent repeat rate parameters are both specified as a number of display frame interrupts, i.e. in units of approximately 20 ms.

If either parameter is given as zero, then the relevant default

```
IU    :  channel in use (queued operation pending)
DE    :  unreported device error
BP    :  bad parameter
```

This call resets (empties) the internal buffer of a keyboard channel, losing any buffered bytes and clearing any unreported data lost warnings. The call acts as an immediate I/O call in a manner analagous to GET BYTE IMMEDIATE.

11.12.6 NOMINATE BREAK-IN KEY

```
Trap Name:              T.KERNEL
Action Value (D0.B):    K.NOMBRK
```

Additional Call Parameters:

```
D1.L  : channel identifier
D2.B  : break-in key code (see Appendices G and H)
D3.W  : event number or -1 (see section 9.4)
D4.L  : target activity identifier
```

Error Returns:

```
NO    : invalid channel identifier or channel not open
IU    : channel in use (queued operation pending)
DE    : unreported device error
NA    : invalid activity identifier
BP    : bad parameter
```

This call enables a break-in key to be nominated for a particular channel. It may only be made by the activity owning the channel or the activity that opened the channel. Any other activity making this call will cause a return with error NO - invalid channel identifier. The call acts as an immediate I/O call in a manner analagous to GET BYTE IMMEDIATE.

In D4.L and D3.W are specified target activity identifier and an event number respectively. Whenever the key code defined in D2.B would otherwise be routed to the channel in question, then the specified event is signalled to the target activity instead. The key code is not routed to the channel. The target activity may be specified as -1, in which case the calling activity will be made the target activity.

If either D2.B or D3.W are specified as -1, the break-in key feature is disabled; in such case the contents of D4.L are ignored.

12. TELEPHONE DEVICE ROUTINE

12.1 General Comments

The control and status reporting of the two telephone lines, built in modem, and handset are multiplexed over a single hardware interface along with the keyboard, tone generation facility and miscellaneous audio path switching functions.

Kernel attempts to simplify this interface by presenting the hardware as a number of independent logical devices. This chapter describes the telephone line logical device and its driver routine. The following chapter describes the modem and communications line facilities.

The telephone line device routine incorporates the major line control function, handset and voice path switching, and control of the LED associated with each line.

Note that direct access to the telephone lines is not possible for normal applications - the facilities described in this chapter can only be used by the trusted OPD activities that comprise the Telephone Handler system [ref 14].

12.2 Device Allocation and Usage Control

The two telephone lines are presented as two separate devices that may be operated independently. It should be noted however that there is only one handset, speech synthesiser and loudspeaker, so that some cooperation will be required between the higher level device handling software.

Only one channel may be opened to each telephone line device.

12.3 Output to the Telephone Channel

The only data written to a telephone channel consists of sequences of control codes which manipulate the hardware, and are described below. For details of the character coding, see Appendix I.

12.3.1 START CALL (START)

This code puts the telephone line associated with the channel into the off-hook condition to answer an incoming call or initiate an outgoing call. (Note that it is currently impossible to tell with any confidence which of these cases has occurred.) A START code must be written before any attempt is made to configure the audio or data paths, or dial on the line. Following this control code, no attempt to send dialling codes should be made for at least one second to allow the dial tone to become established. (This will be the normal user action in manually dialled calls anyway.)

acknowledged, the number of DIALX codes routed to the input side of the channel is indeterminate.

The telephone handling software does not need to know whether the type of dialling needed is loop-disconnect or DTMF, except that DIAL* and DIAL# codes must not be sent to a loop-disconnect line.

Note that there should be a delay of at least one second between sending a START code and sending the first digit to be dialled, to enable dial tone to become established.

12.3.9 SEND DTMF DATA (DTMFh)

These codes send the relevant DTMF tones for a given code to the line associated with the channel ($0 \leq h \leq \$F$). The buffering and acknowledgement reporting is the same as for dialling normal digits.

For dialling, the DIALx codes should be used, and on DTMF lines may be used as an alternative to the DTMF codes. On loop disconnect lines, DTMFh codes can be used to generate the DTMF tones once a call has been established. Note that these codes are not implemented by all firmware versions of the Telephony Module [ref 3].

12.3.10 SWITCH LED ON (LED)
12.3.11 SWITCH LED OFF (XLED)

These codes switch the LED associated with the relevant line on or off, as required. Because of the requirements of ULA2 [ref 2], there may be a perceptible delay before the LED state is physically changed (and if a further change is requested almost immediately the first change may not be visible at all). This situation only occurs when bytes are actually being transmitted across the printer interface.

12.3.12 RECALL (RCAL)

There are two basic forms of Recall, namely Earthed Recall and Timed Break Recall (also known as hook-flash recall). This code initiates the relevant recall sequence configured for the line associated with the channel. (The means by which a line is configured is given in Appendix I.)

In the case of Earthed Recall, the recall condition is established, and no further action taken. In the case of a Timed Break Recall, a "mute - timed break - unmute" sequence is initiated, as described by the permanent store Recall configuration entry for the line.

Note that while a Timed Break Recall sequence is in progress, any RCAL or XRCAL code (see section 12.3.13) on either channel is ignored; any DIALx or DTMFh code on the channel associated with the line is ignored: and the action of any MUTE XMUTE

Until either of these codes is received, the only code that can be written to the channel is XMODEM, to shut down the modem prior to any further telephone line manipulation.

12.3.18 STOP MODEM (XMODEM)

This code stops any data call connection that is in progress or in the process of being established, to enable further operations on the telephone line to take place (such as the continuation of manual dialling). If a previous MODEM code has successfully established the communications channel and allowed modem I/O to proceed, then any open modem channel is forcibly closed.

12.4 Input from the Telephone Channel

Input data from the telephone channel can comprise both control codes arising from changes of hardware state, and keyboard input. For further details of telephone input arising from the keyboard, see Chapter 11.

The control codes that may appear as input from a telephone channel as a result of telephone or modem hardware state changes are described below. The input data arising from the keyboard no has inherent meaning ascribed by Kernel. For details of the character coding see Appendix I.

12.4.1 HANDSET UP (HUP)
12.4.2 HANDSET DOWN (HDOWN)

These codes indicate that the handset state has changed, and are reported on the channel associated with the line to which the handset is currently switched.

12.4.3 RINGING STARTED (RING)
12.4.4 RINGING STOPPED (XRING)

These codes indicate that ringing on the line associated with the channel has been detected (RING) or stopped (XRING). Note that the sequence and timing of these codes matches the actual ringing pattern generated at the exchange (apart from any firmware and software propagation delays).

12.4.5 DIGIT(S) DIALLED (DIALX)
12.4.6 DTMF DATA SENT (DTMFX)
12.4.7 RECALL FINISHED (RCALX)

These codes are returned on the input side of the channel as a dialling or recall sequence is completed. See the DIALh, DTMFx, RECALL and XRCAL codes above.

12.4.8 DATA CALL ESTABLISHED (DCE)

This code is returned as input when a data call initiated by a

This call is not implemented for telephone channels and when made by the channel owner or opener it will always return error NI - operation not implemented. The effect of any subsequent I/O call on the channel is undefined.

12.7.3 GET BYTE IMMEDIATE

Call Parameters and Error Returns:

see section 9.7.3

This call behaves as described in section 9.7.3, but the following points should be noted:

a) Sufficient channel buffering is provided such that the Data Lost device warnings should never occur.

b) Any call of PUT BYTE IMMEDIATE for the channel will cancel any event notification requested by a previous GET BYTE IMMEDIATE call.

12.7.4 PUT BYTE IMMEDIATE

Call Parameters and Error Returns:

see section 9.7.4

This call behaves as described in section 9.7.4, but the following points should be noted:

a) A certain amount of channel buffering is provided to handle the situation where the telephone module cannot immediately accept a (further) control code.

b) Provided that a subsequent DIALx or DTMFh code is never put to the channel while a previous one remains unacknowledged on the input side, then error NB - no byte passed - is never returned from this operation.

12.7.5 QUEUED GET STRING
12.7.6 QUEUED GET STRING WITH TERMINATOR
12.7.7 QUEUED PUT STRING

These calls all return error NI - operation not implemented.

12.7.8 GIVE CHANNEL STATUS

Call Parameters and Error Returns:

see section 9.7.8

No device errors are returned from the telephone driver.

The only device warning that may be returned is Data Lost, and

13. MODEM DEVICE ROUTINE

13.1 General Comments

The telephone module incorporates a built-in modem capable of supporting a variety of different data formats and protocols [ref 3].

In particular the modem supports communication in any of the following modes:

a) half duplex
b) half duplex with back channel enabled
c) full duplex

In all of these modes, Kernel presents the modem as two separate channels (one for input and one for output). These two channels may be operated more or less independently, although true half duplex operation will obviously require a degree of coordination between the two to be exercised.

13.2 Device Allocation and Usage Control

A maximum of two channels may be opened to the modem at one time; if two channels are used then one must be an input channel and one an output channel.

The two modem channels can be opened by any activity, subject to the restriction that the target activity of the second open must be the guardian of the activity associated with the first channel opened, or share the same guardian as that activity. [Note that this restriction allows both channels to be opened by the same activity.]

The OPEN CHANNEL call may be made at any time and returns immediately. The call specifies the modem set up and configuration required and how it is to be started, but this is not actioned in any way until a phone line is switched into data mode (after the relevant phone line set up has been completed by telephone handler).

Following an OPEN CHANNEL call, the activity associated with the channel may make read and write I/O calls, but no data will actually be transferred until the phone line is switched to data mode, and the data call established. (For further information see Chapter 12 and [ref 14].)

13.3 Data Formats

The input and output characters are treated by Kernel as transparent binary data, with 7 or 8 significant bits and parity as specified in the set up parameters.

directly the number of bytes lost in this manner, nor the position of the missing bytes.

13.5.2 RECEIVE BUFFER OVERFLOW

It is possible that for some reason Kernel itself is unable to keep up with the telephone module/modem causing an 'Rx buffer overflow' status to be sent from the telephone module [ref 3]. While this should never happen under normal circumstances, any such occurrence is passed to the input channel via a device warning flag, in the same manner as for normal data lost. It is not possible to determine directly the number of bytes lost in this manner, nor the position of the missing bytes.

13.5.3 CARRIER FAILURE

At various times, depending on the modem setup and configuration details, the telephony module can return a Data Channel Failure status [ref 3] indicating that carrier has been lost. When this occurs before the modem has been properly started, the telephone channel is informed by means of an XDCE code (see section 12.4.8). Once the modem has been set up and the data call/line properly established, then a Data Channel Failure status from the telephone module is handled rather differently, as follows.

Whenever such a status is returned from the telephony module, a device warning flag is set in the channel flag byte to indicate the error. (This flag is cleared by a GIVE CHANNEL STATUS or CANCEL I/O TRANSFER call.) Kernel keeps a count of consecutive channel failures, which is reset whenever anything happens on the interface that implies carrier is present. If the count exceeds a value that corresponds to approximately 1 second of carrier loss, any outstanding modem transfer is terminated with device error CF - communications channel failure.

Where modem setup and telephony firmware permit, Kernel checks for carrier remaining every 100 ms (when no transfers are outstanding), and will set device error CF - carrier failure - if carrier disappears for approximately 1 second. For further details see [ref 3].

13.6 Modem Driver Serial I/O Kernel Calls

13.6.1 OPEN CHANNEL

Call Parameters: see section 9.7.1

Logical Device Number: DV.MODEM

Additional Call Parameters:

A1 : address of further information (see below)

This call opens a modem channel for input or output. The call

Certain Telephone Handler actions can force a close operation on the modem channel(s). Further details of the interactions between the modem device routine, telephone device routine and Telephone Handler are given in Chapter 12 and [ref 14].

13.6.3 GET BYTE IMMEDIATE

Call Parameters and Error Returns:

see section 9.7.3

If the channel is an input channel, this call behaves as described in section 9.7.3

If the channel is an output channel, this call returns error NI - operation not implemented.

13.6.4 PUT BYTE IMMEDIATE

Call Parameters and Error Returns:

see section 9.7.4

If the channel is an output channel, this call behaves as described in section 9.7.4.

If the channel is an input channel, this call returns error NI - operation not implemented.

13.6.5 QUEUED GET STRING

Call Parameters and Error Returns:

see section 9.7.5

If the channel is an input channel, this call behaves as described in section 9.7.5.

If the channel is an output channel, this call returns error NI - operation not implemented.

13.6.6 QUEUED GET STRING WITH TERMINATOR

Call Parameters and Error Returns:

see section 9.7.6

If the channel is an input channel, this call behaves as described in section 9.7.6.

If the channel is an output channel, this call returns error NI - operation not implemented.

13.6.10 CHANGE CHANNEL OWNERSHIP

Call Parameters and Error Returns:

see section 9.7.10

13.7 Additional Modem Driver Kernel Calls

13.7.1 TURN LINE ROUND

Trap Name:	T.KERNEL
Action Value (D0.B):	K.MODEMTLR

Additional Call Parameters:

D1.L	:	channel identifier
D2.W	:	new line state

Error Returns:

NO	:	invalid channel identifier or channel not open
IU	:	channel in use (queued transfer in progress)
DE	:	unreported device error
BP	:	bad parameter

This call enables the line to be turned round and communication to proceed in a different direction from that currently set.

If two channels have been opened to the modem then either one may be specified on this call. If a queued I/O operation is in progress on that channel, then an immediate return is made with error IU - channel in use. If the channel specified in the call has an unreported device error, then an immediate return is made with error DE - unreported device error. No check is made for any outstanding I/O operation or unreported device error on the other channel.

The new line state is specified in D2.W as follows:

1	:	put line in half duplex transmit mode
2	:	put line in half duplex receive mode

If the modem hardware is configured for full duplex, this call has no effect. If the modem is configured with back channel enabled and the new direction is different from that already established then the effect is to change the two line speeds round. If the modem is configured for half duplex without a back channel, and the new direction is different from that already established then I/O will only proceed in one direction and that direction is reversed by this call.

14 PRINTER DEVICE ROUTINE

14.1 Basic Device Characteristics

The OPD is capable of supporting a variety of different printers; these are fully described in [ref 8]. The Kernel I/O calls provide a transparent interface to the printer hardware. Applications are normally expected to use the Printer Manager facilities, which handle the differences between the various models of printer [ref 16].

The graphic/printable characters supported include the basic OPD character set described in Appendix E. Control codes and additional graphic characters are fully described in [ref 8].

14.2 Device Allocation and Usage Control

Only one channel may be open to the printer at any given time, and this channel is a direct connection to the printer hardware. Kernel does not provide any offline spooling capability.

14.3 Channel and Device Buffering

Characters destined for the printer are written directly to the hardware, and are not internally buffered in the channel. The device itself, however, may be capable of buffering a significant number of characters.

14.4 Device Status Information

The only status information available from the hardware is a 'device busy' status, which means a variety of things e.g. the printer being switched off, out of paper or just printing the current buffer contents. Kernel does not attempt to assign any extra meaning to the device busy status. It is assumed that activities using the printer will use a sensible timeout mechanism and inform the user if the printer becomes non-functional.

Note that access to the microdrives and printer involves the use of common hardware, and hence printer I/O may on occasions take a significant time to complete, because no bytes can be transmitted to the printer while a microdrive is spinning. Providing that long enough timeouts are used, then this hardware sharing is transparent at the Kernel interface level.

14.5 Printer Driver Serial I/O Kernel Calls

14.5.1 OPEN CHANNEL

Call Parameters and Error Returns:

see section 9.7.1

status, and generate the requested event if and when the printer becomes able to accept a byte again.

14.5.5 QUEUED GET STRING
14.5.6 QUEUED GET STRING WITH TERMINATOR

These calls always return error NI - operation not implemented.

14.5.7 QUEUED PUT STRING

Call Parameters and Error Returns:

see section 9.7.7

14.5.8 GIVE CHANNEL STATUS

Call Parameters and Error Returns:

see section 9.7.8

This call behaves as described in section 9.7.8. No device errors or warnings are returned from the printer device routine.

14.5.9 CANCEL I/O TRANSFER

Call Parameters and Error Returns:

see section 9.7.9

This call behaves as described in section 9.7.9. Note that this call only cancels the sending of further data to the printer hardware. There is no means provided for cancelling the effect of any data that may remain buffered in the printer itself.

14.5.10 CHANGE CHANNEL OWNERSHIP

Call Parameters and Error Returns:

see section 9.7.10

15.2.1 CARTRIDGE FORMATTING

Before it can be used by the filing system, a microdrive cartridge must be formatted. This process determines the exact size of the cartridge, creating the fixed length sectors and checking for flawed areas on the tape. Various items of red tape are written to the cartridge during this process, in particular a volume name, which allows subsequent identification of the volume by the filing system.

15.2.2 VOLUME NAMES

Each microdrive cartridge has a volume name, which is written to the cartridge when it is formatted. The volume name is chosen by the user, and it is strongly recommended that each cartridge be given a different volume name. The effects that might occur if two cartridges with the same name are mounted are undefined.

Before any read or write operation is performed on a cartridge, a check is made that the name on the cartridge is the one expected. If each cartridge has a different name, this check provides a measure of protection against a cartridge being removed from a microdrive and replaced with another. (The check does not extend to the volume uniqueness field in the Volume Description Record.)

At the Kernel interface level, a volume name consists of up to 8 graphic characters chosen from the basic OPD character set (see Appendix E). The characters can be freely chosen as Kernel ascribes no meaning or structure to volume names; other software regimes may force a subset or impose conventions as required. Volume names are always stored on a cartridge as supplied, but any search for volume name equality treats upper and lower case alphabetic characters as being identical.

15.2.3 CATALOGUE FILES

Each microdrive volume has two catalogue files, which are created when the volume is formatted. The two files are identical and duplicated for reasons of resilience and speed of access. The catalogues hold information about all the user files on the volume, including the two catalogues themselves. Catalogue files are described in more detail below.

15.2.4 CARTRIDGE STATISTICS

It is anticipated that the largest threat to data security will come from deterioration of the microdrive cartridge with use. To prevent the performance of the microdrive filing system deteriorating to the point of unreliability, Kernel keeps some statistics on cartridge usage and performance.

These statistics can be examined as desired, and in particular may be used by the OPD software to issue recommendations to the

in Appendix J.

As well as its documentary rôle, the file type can be used by Kernel to control (to some extent) the file space allocation and block access algorithm.

Each file type is effectively extended by the file type qualifier, which is a numerically coded value used in conjunction with the file type. The range of values and semantics of file qualifiers depend on the file type, and are beyond the scope of this document. Kernel ascribes no meaning to file qualifiers.

15.3.4 FILE NUMBERS

Each file on a volume has a file number. This is a value in the range 1 to 242, and is used within the filing system to identify a file. The file number is mainly of use to those applications who wish to optimise their microdrive access by examining the sector allocation map of the volume. (See sections 15.5 and 15.6 for further details.)

It is not possible to access a file directly by file number.

15.3.5 UNIQUENESS OF FILE NAMES

A microdrive volume may not contain more than one file with the same file name. However, files with the same name may occur on different volumes. If two such volumes are mounted at the same time, then the actual file used depends on the parameters passed to the open call. Note that upper and lower case alphabetic letters are regarded as being the same so far as volume names and file names are concerned.

It is expected that higher levels of software will use part of the filename as an extension to indicate the logical type of file, in a manner similar to other microprocessor operating systems.

15.3.6 SECTOR ALLOCATION AND ERROR RECOVERY

Kernel assumes that microdrive files are dense, i.e. that all the within the addressing range of the file will eventually hold blocks user data. To this end, (along with the performance considerations described in section 15.6), whenever a file is created or extended, sectors are allocated in the sector map to hold all the blocks in the file. It is assumed that all the blocks so allocated will be written to (normally before the file is closed for the first time); this assumption is not validated by Kernel in any way, nor is any attempt made to initialise unused blocks.

This poses no real problems in normal circumstances, except that any attempt to read a block before it has been written to will return garbage data in the user's buffer.

The volume description record is read into RAM by Kernel the first time the volume is accessed, and kept there; the cartridge sectors containing the volume description record are updated automatically (and fairly frequently) even if the cartridge is 'write protected' (see section 15.2.4 above).

The format of a Volume Description Record is given in Appendix J.

15.4.3 FILE DESCRIPTION BLOCKS

The second block of a catalogue file is a file description block (as are any further blocks). Each file description block contains between 0 and 11 file description entries.

Each file description entry contains information about a particular file on the volume, and is used to gain and control access to the file.

Each file on the volume, including each of the two catalogue files, has a File Description Entry.

The format of a File Description Entry is given in Appendix J.

15.4.4 CATALOGUE FILE ACCESS

As the catalogue files are essentially normal microdrive files they can be accessed as such from any application that wishes to read the contents. Catalogue files cannot, however be written to in this manner as the files are 'write protected' as far as normal I/O calls are concerned.

The contents of the catalogue are updated and altered implicitly by most microdrive access calls; a certain amount of explicit updating is also provided for microdrive utility programs and similar such applications.

Below the level of the Kernel interface, access to the catalogue files is slightly different, in that regardless of which catalogue file is actually opened, in general Kernel will use the first relevant block of data that becomes available. Applications should normally restrict their access to !CAT1, to maintain future code compatibility with other filestore volumes.

Note that while an application has one or other of the catalogue files open for reading, no changes can be made to the catalogue file, and in particular, no other files can be created or opened for writing.

15.4.5 CATALOGUE ERROR RECOVERY

When a catalogue file is directly accessed by an application via the Kernel I/O calls then, while the transfer is being set up, Kernel will arrange to read the first relevant duplicate block that is expected under the read head. Should this block fail to

15.5.2 FILE EXTENSION

If, when a file is being written to, a file address higher than the current end of the file is given, the file is extended (space permitting) to the new size, and further sectors allocated. The new sectors are allocated in the same manner as at file creation if possible, but if a file is extended piecemeal the allocated sectors are likely to become more and more randomly scattered, and hence cause performance degradation.

Such piecemeal extension should be avoided if possible, particularly when a lower limit on the eventual file size can be predicted with any degree of accuracy.

15.5.3 ENTIRE FILE OPERATIONS

Where possible (this obviously depends on the file size) it is likely to be advantageous if the whole file can be read into RAM or written from memory in one operation. Kernel provides calls to accomplish this.

15.5.4 MULTI BLOCK TRANSFERS

The Kernel block access interface allows an application to ask for several blocks in a file to be read or written 'simultaneously', and where possible, the reads will be completed within one revolution of the tape and the writes within two revolutions.

15.5.5 PERFORMANCE ESTIMATION

As well as the above considerations, a number of other factors should be borne in mind when estimating the perceived performance:

a) there is a mandatory read-after-write check performed on all write operations which requires an extra revolution of the tape after the block is written.

b) opening a file for writing will be a slow process because of the need to update the catalogue files (writing the update to the volume and then verifying the update).

c) at any time Kernel may elect to suspend the microdrive software to handle some short term crisis. This will sometimes result in a required sector passing the heads unnoticed by the microdrive software, resulting in a minimum 7 second delay before it comes round again.

15.6 Filestore Slaving

Filestore slaving is a technique whereby some or all of the data blocks in a file are also held in RAM. Application program I/O requests can then be satisfied by store-to-store moves, without

open for reading or writing, then a failure is reported.

d) Open New for Writing: the file must not exist; a new file is created and opened for writing. If the file is present, a failure is reported.

e) Open for Writing: if the file is present, and not open for reading or writing, then it is first deleted, then recreated at the specified size.

15.7.2 CLOSE CHANNEL

The call to close a channel to a microdrive channel operates in exactly the same manner as the serial I/O call described in section 9.7.2. Note that a close operation always cancels any outstanding I/O first, and hence should not normally be made until all write I/O operations on the file have completed.

Microdrive file channels should always be closed when the application has finished with the file, as this may free substantial resources, and allow the file to be reopened in a different mode.

15.7.3 QUEUED BLOCK ACCESS

This call is analagous to the QUEUED GET STRING and QUEUED PUT STRING operations described in section 9.7, in that it initiates I/O to the file associated with the channel, and signals termination of that I/O by causing an event in the normal manner.

However, this call differs in that it allows several input and/or output operations to be specified in one call; these operations are performed as 'simultaneously' as possible, and termination is only signalled when all the operations specified in a particular call have completed.

Kernel aims to complete all the read requests in one revolution. Write requests are subject to a mandatory read-after-write check and hence the aim is to complete all the write requests in two revolutions.

A number of factors control the actual number of revolutions required, or the number of transfers completed on any particular revolution. These include error recovery, sector placement, RAM availability, and activity in the rest of the system.

15.7.4 LOAD FILE

This call is similar to the QUEUED BLOCK ACCESS call above, and allows an entire file to be read/loaded in one operation, to a contiguous area of RAM.

15.8.5 GIVE DRIVE INFORMATION

This call returns a certain amount of information concerning the current state of sector allocation and positioning, to enable applications to optimise their microdrive accesses.

15.9 Microdrive Block Access I/O Kernel Calls

15.9.1 OPEN CHANNEL

Call Parameters and Error Returns:

see section 9.7.1

Logical Device No: DV.FILE

Additional Call Parameter:

A1 : address of further information

This call opens a channel to a microdrive file, exactly as for the Serial I/O OPEN CHANNEL call described in section 9.7.1. The call will not normally complete immediately and unless any failures are detected at the time of the call, it will return with error NC - operation not complete.
A1 contains the address of further information, formatted as follows:-

0	m	d	s	r	t	q						
6	v	v	v	v	v	v	v	v				
14	f	f	f	f	f	f	f	f	f	f	f	f
26	z											

m : mode of open
0 - check for presence
1 - open for reading
2 - open for overwriting
3 - open new for writing
4 - open for writing

d : drive number
0 - either
1 - drive 1 (right hand)
2 - drive 2 (left hand)

s : file size in blocks (0 to 254)

r : slaving request information
0 - don't mind
1 - do slave if possible
2 - slave entire file if possible
3 - do not slave

drive number	volume name	access mode	action taken
n	given	all	drive n checked for correct volume
0	given	all	drive 1 then drive 2 checked for correct volume
n	-	all	volume on drive 'n' used
0	-	0,1	drive 1 then drive 2 checked for correct filename
0	-	2,3,4	error BP returned

When the open finally terminates, the requested event (if any) is signalled, and a GIVE CHANNEL STATUS call can be made to determine the success or failure of the call. The device errors that may be reported following a delayed open are given in section 15.9.7.

Certain failures are detected before any I/O is initiated, in which case the OPEN CHANNEL call makes an immediate return with an error code other than NC - not complete. The errors that may be returned at this stage are as follows:-

OM	:	out of memory (unable to create CCA)
NA	:	invalid target activity identifier
MD	:	special CCA still in use [z = $FF only]
BP	:	bad parameter

15.9.2 CLOSE CHANNEL

Call Parameters and Error Returns:-

see section 9.7.2

This call closes a microdrive channel and frees resources in exactly the same manner as the Serial I/O CLOSE CHANNEL call described in section 9.7.2. Note that access to the microdrives and use of memory resources such as the channel control area may continue for some time after this call, even though an immediate return is made. However, any attempt to perform further I/O operations on the channel will of course fail error NO - channel not open.

15.9.3 QUEUED BLOCK ACCESS

Trap Name :	T.KERNEL
Action Value (D0.B):	K.QBLOCK

Additional Call Parameters:

0 - do nothing
1 - read block
2 - write block

b : byte address within file

a : store/buffer address

l : transfer length

The 'do nothing' option in the f field allows for fixed length transfer control blocks if desired.

On write operations, the byte address within the file must be a multiple of 512 bytes, and the transfer length field must be 512 bytes. The file must have been opened in one of the modes that allows writing.

On read transfers any byte address and length may be chosen provided that the whole slice remains within one 512 byte block (there is no additional buffering implied by this facility).

At the time the transfer is initiated by the call, the contents of the s field are ignored.

When the call is made, the transfer control block is scanned, and the block transfer status field for each entry is set positive and non zero, indicating that the particular block transfer is in progress. As each block transfer terminates successfully, then the s field in the relevant entry is set to zero. If any block transfer fails for any reason then a device error code (which will be negative) is set in the relevant s field.

When all the block transfers have completed, then the channel/device status field is set to zero if all the block transfers were successful, or to error BF - block transfer failure - if one or more failed for any reason. A termination event is then signalled if required, as for serial I/O queued operations.

The block transfer failures that can be reported throughout the transfer are as follows:

PF - block check fail on read - after (several) retries the required block could not be read from the file

OS - out of sectors - a write operation specified a byte address greater than the current end of file address, and there were insufficient free sectors on the microdrive to allocate the extra block(s); or there is no sector to satisfy or complete error recovery (see section 15.3.6); or, following volume recovery because of an improperly closed file, a read operation specified a block not written to the file (see section 15.3.6)

This call queues an I/O transfer to read a logically contiguous number of bytes from a file into a contiguous area of RAM. The call returns immediately, and behaves in a manner exactly analogous to the QUEUED GET STRING call described in section 9.7.5.

The file contents, from the specified start address for as many bytes as specified, are read directly into the caller's buffer. The channel to the file must have been opened in any of the modes except 'check for existence'.

The various blocks required from the file are read asynchronously as they become available, and the I/O terminates in the normal manner when all the requested bytes have been read. Kernel attempts to complete this call in one revolution.

The start address passed in D4.L is the byte address within the file, and must be a multiple of 512.

The length specified in D2.L need not encompass the whole file, nor need it be a multiple of 512 bytes. If the length specified combined with the start address is greater than the length of the file, then no error is signalled, and the remainder of the user's buffer left unaltered.

15.9.5 QUEUED SAVE FILE

Trap Name:	T.KERNEL
Action Value (DO.B):	K.SAVEFILE

Additional Call Parameters:

D1.L	:	channel identifier
D2.L	:	buffer/file length (in bytes)
D3.W	:	event number or -1 (see section 6.4)
D4.L	:	start address in file
A1	:	address of buffer

Error Returns:

NO	:	invalid channel identifier or channel not open
IU	:	channel in use (queued operation pending)
NI	:	operation not implemented on this channel
DE	:	unreported device error
OS	:	out of sectors
BP	:	bad parameter

This call queues an I/O transfer to write a contiguous number of bytes from memory to a file. The call returns immediately, and behaves in a manner exactly analgous to the QUEUED PUT STRING call described in section 9.7.7.

The contents of memory are written, for as many bytes as

create or extend the file or the catalogue)

UF - unformatted volume (the cartridge on the specified drive is not formatted for the OPD microdrive system; or no cartridge is mounted on the specified drive; or the volume is use has been removed and replaced by another)

CO - catalogue already open for reading (when attempting an operation which implies that Kernel must open the catalogue for writing, or when attempting to open the catalogue itself for writing)

WP - write protect fail (an attempt to open a file for writing has been made when the cartridge is write protected)

US - volume unusable (both duplicates of a catalogue file block have become unreadable)

On an open to check for a file's existence, errors IU and CO will not be returned.

Following a failed open, a CLOSE CHANNEL (or DESTROY FILE) call should be made, and any other I/O call on the channel will immediately return with error NI - operation not implemented on this channel.

The only device error that may be returned following termination of a QUEUED BLOCK ACCESS call is error BF - block access failure - when one or more supplementary device error codes will be found in the transfer control block, as described above.

The device errors that may be reported following a QUEUED LOAD FILE or QUEUED SAVE FILE operation are those described as 'block transfer failures' in section 15.9.3. If a failure is detected during these operations, the first block transfer failure is reported as a device error, and the remainder of the transfer cancelled.

The device errors that may be reported following a QUEUED CHANGE FILE DESCRIPTION operation are any of the errors described above, although some errors may only occur if the cartridge is surreptitiously replaced with a another of the same volume name.

If any access to a volume is made after the volume has erroneously been removed from the drive, or suffered a major failure (errors UF and US), then the error is reported in the channel in the normal way, and the channel flagged such that any subsequent attempts to perform I/O on the channel will immediately return with error NI - operation not implemented on this channel.

IU	:	channel in use (queued operation pending)
DE	:	unreported device error
NI	:	operation not implemented on this channel
BP	:	bad parameter

This call returns information about the file associated with the channel. It returns immediately but may only be made when no I/O is in progress on the channel. If a queued operation is in progress on the channel, return is made with error IU - channel in use.

A1 contains the address of the caller's buffer, into which is copied the following information:-

byte 0	:	drive number
bytes 1 to 8	:	volume name
bytes 9 to 52	:	file description entry

The number of bytes to be copied is passed in D2.W. If this is greater than 53, only 53 bytes will be copied; if less, only the specified number will be copied.

The format of a file description entry is given in Appendix J.

Note that if the channel identifier passed to this call is derived from a failed or cancelled open, then an immediate return is made with error NI - operation not implemented on this channel.

15.10.2 QUEUED CHANGE FILE DESCRIPTION

Trap Name:	T.KERNEL
Action Value (D0.B):	K.QCFILE

Additional Call Parameters:

D1.L	:	channel identifier
D2.L	:	length of buffer
D3.W	:	event number or -1 (see section 6.4)
A1	:	address of buffer

Error Returns:

NO	:	invalid channel identifier or channel not open
IU	:	channel in use (queued operation pending)
DE	:	unreported device error
NI	:	operation not implemented on this channel
BP	:	bad parameter

This call enables certain alterations to be made to the file description entry of the file associated with the channel. The file must have been opened in one of the modes that allows writing. If the file is not so opened, an immediate return is made with error NI - operation not implemented on this channel.

Additional Call Parameters:

D1.W	:	drive number
D2.W	:	must be zero
A1	:	new volume name address
A2	:	old volume name address or zero

Return Parameter:

D0.B	:	count of free sectors

Error Returns:

NC	:	operation not complete
IU	:	volume in use
NV	:	volume not found
WP	:	volume is write protected
US	:	cartridge is unusable
MD	:	other microdrive is in use
BP	:	bad parameter

This call is only to be used by the microdrive formatting utility. It formats a new or existing volume, thus totally destroying any previous contents.

The drive number containing the cartridge to be formatted is specified in D1. A1 contains the address of an 8 byte area containing the new volume name. If the cartridge is a previously unformatted one, A2 must be zero. If the cartridge has previously been formatted, then A2 must contain the address of an 8 byte area containing the existing volume name. If the cartridge mounted on the specified drive contains a different volume, return is made with error NV - volume not found.

If any files are open on the specified drive/volume then return is made with error IU - drive in use. If the other drive is spinning then return is made with error MD - other microdrive in use.

Once the formatting has started, interrupts are inhibited, and the whole machine will apparently go dead for a period of 45 seconds or more, with no other software being allowed a share of the CPU. The format process periodically polls the interrupt register, and if any interrupt from the telephone module is noticed (i.e. keyboard depression, telephone line activity, or modem data arriving) then the FORMAT VOLUME call is immediately terminated with error NC - operation not complete.

Once the physical level formatting is complete, new, duplicate, empty catalogue files are created, and the total number of free sectors is returned in D0.B. If the number of usable sectors found by the physical level formatting is less than 10, no catalogue blocks are written and return is made with error US -

16. SPEECH SYNTHESISER DEVICE ROUTINE

16.1 General Comments

The OPD hardware includes a Texas Instruments TMS 5220C Voice Synthesis Processor, and associated TMS 6100 ROM containing the speech data [ref 10,11].

The output from the synthesiser may be directed to either of the telephone lines, or to the loudspeaker.

The ROM containing speech data also contains textual descriptions of the phrases therein, and facilities to read this data are provided.

16.2 Device Allocation and Usage Control

Only one activity may allocate the device at any given time, and the destination of the speech output (if any) is given at OPEN CHANNEL call time. If this destination is one or other of the telephone lines, then the calling activity must already have a channel open to that telephone line.

Further, if a channel to the synthesiser is already open in voice loopback mode (i.e. monitored on the loudspeaker) or read only mode, and an OPEN CHANNEL call is made to connect the synthesiser to a phone line, a CLOSE CHANNEL operation is forced on the currently open channel, to allow the telephone handler priority access (with the effects described in section 9.6.2).

When the synthesiser is used in conjunction with the phone lines, the Kernel mechanism assumes a certain amount of cooperation from the telephone handling system software, as described below.

16.3 Serial I/O Operations

The serial I/O mechanism is used to pass both control and data information from the synthesiser. On 'write' operations, both control information and data can be passed; on 'read' operations, only data is passed.

16.3.1 WRITE OPERATIONS

The following control functions are initiated by writing the specified escape sequences:-

16.3.1.1 Speak from ROM (SPEAK/address)

This causes the synthesiser to make the noises described by the coded data starting at the specified Speech ROM address

16.3.1.2 Speak External (SPKEXT)

This control function is followed by direct speech data

16.6 Speech Synthesiser Driver Serial I/O Kernel Calls

16.6.1 OPEN CHANNEL

Call Parameters and Error Returns:

see section 9.7.1

Logical Device Number : DV.SYNTH

Additional Call Parameters:

A1 : address of further information (or zero)

This call opens a channel to the synthesiser. The call always completes immediately, and error NC - operation not complete - will not occur on synthesiser OPEN CHANNEL calls.

If the contents of A1 are zero, then the synthesiser is made available for reading only, and no sounds will be generated. If the contents of A1 are non-zero, then A1 contains the address of a single byte which specifies the destination of the speech output, as follows :-

0	:	null (read only)
1	:	loudspeaker
2	:	telephone line

Only one activity can open a channel to the synthesiser at once. If the telephone line is specified as the output of the generated speech, the activity opening the channel must be a System Activity; the target activity may be of any variety. If such an OPEN CHANNEL call is made when a channel to the synthesiser is already open for reading or loudspeaker use, then a CLOSE CHANNEL operation is forced on that channel before allowing the priority OPEN CHANNEL to complete successfully. If such a call is made when a channel is already open for telephone line use, then return is made with error IU - device in use.

If a channel to the synthesiser is opened for loudspeaker loopback or read only use, the calling and target activities may be of any variety, but must be prepared to get error NO - channel not open - on any serial I/O call in the event of a priority open operation for telephone line usage.

When the destination of the speech output is the loudspeaker, the audio path is set up at the time of the OPEN CHANNEL call. When the destination is one or other of the telephone lines, then Kernel assumes that the telphone handling software will have connected the synthesiser to the relevant line (see Chapter 12) before proceeding with any I/O operations.

remaining data in the caller's buffer is assumed to be speech data, and this will all be written to the device. Note that if a number of QUEUED PUT STRING operations are being used to speak user supplied data, then each I/O call should start with the Speak External command.

16.6.8 GIVE CHANNEL STATUS

Call Parameters and Error Returns:

see section 9.7.8

This call behaves as described in section 9.7.8.

No device errors or warnings are reported by the synthesiser device routine. The only error condition is that which occurs when an OPEN CHANNEL call for connection to the phone line overrides an existing channel (as described in section 16.7.1). In this case, a GIVE CHANNEL STATUS call on the usurped channel will return with error NO - channel not open.

16.6.9 CANCEL I/O TRANSFER

Call Parameters and Error Returns:

see section 9.7.9

This call behaves as described in sections 9.7.9 and 16.6.8.

16.6.10 CHANGE CHANNEL OWNERSHIP

Call Parameters and Error Returns:

see section 9.7.10

Logical Device Number : DV.PIPE

Additional Call Parameter :

A1 : address of further information (see below)

This call opens a channel to a named pipe; the call always completes immediately, and error NC - operation not complete - will not be returned.

A1 contains the address of a word aligned field (there is no default); the format of this field is as follows:

b	b	c	z	n	n	n	n	n	n	n	n

b : input channel buffer size (1 - 255)
c : channel directionality
0 = input channel
1 = output channel
z : must be zero
n : pipe name (8 bytes)

The call searches for another pipe channel with the same name as that specified in the 'n' field. If such a channel is found, and the named pipe is already complete, or the channel has the same directionality as that specified by the 'c' field, then return is made with error IU - device in use. If such a channel is found with the opposite directionality to that specified in the 'c' field and the pipe is incomplete, then the pipe is completed and a successful return made by the call.

If no channel with the specified pipe name is found, a channel is set up (with the relevant name) and a successful return made. Obviously, no I/O operations will terminate until the pipe is completed by the second, matching, OPEN CHANNEL call; if it is necessary to wait for pipe completion before issuing any I/O, completion can be determined by examining the channel flags returned from a GIVE CHANNEL STATUS or CANCEL I/O TRANSFER call if required.

For input channels, the 'b' field of the additional information area defines the amount of internal buffering required, in bytes. For output channels, the 'b' field is ignored.

It is tentatively suggested that pipe names follow the project naming standards, even though any 8 byte binary value suffices for Kernel.

17.5.7 QUEUED PUT STRING

Call Parameters and Error Returns:

see section 9.7.7

If the channel is an output channel this call behaves as described in section 9.7.7. If the channel is an input channel this call returns error NI - operation not implemented.

17.5.8 GIVE CHANNEL STATUS

Call Parameters and Error Returns:

see section 9.7.8

This call behaves as described in section 9.7.8.

No device errors or warnings are reported by the pipe device routine.

For a pipe channel, the channel and device flag bits have the following significance:-

```
bit 7 = 0   pipe is incomplete
        1   pipe is complete
```

17.5.9 CANCEL I/O TRANSFER

Call Parameters and Error Returns:

see section 9.7.9

This call behaves as described in section 9.7.9.

This call cancels any outstanding I/O on the specified channel; the other channel of the pipe is not notified.

No bytes buffered internally in the channel are lost by this call.

17.5.10 CHANGE CHANNEL OWNERSHIP

Call Parameters and Error Returns:

see section 9.7.10

Error Returns:

NO : invalid channel identifier or channel not open
BP : bad parameter

This call sets or resets the timer associated with the given channel identifier. If any previous timer call on the channel is active/outstanding, then it is cancelled, and the new requirement set up.

The parameter passed in D2.W specifies the period that will elapse before the event associated with the call is caused. Kernel will cause this event after the one second Real Time Clock hardware has been observed to change the number of times specified in D2. The delay is hence between n-1 and n seconds (where n is the value in D2). [Note that the accuracy of the delay will suffer if microdrive formatting is performed.]

If the value passed in D2 is zero it is ignored. If positive, the event is caused once. If negative, the absolute value specifies the delay, but the event will continue to be generated each time that interval expires.

The parameter passed in D3 is the number of the event that is to be caused when the interval expires. A value of -1 may be passed to cancel any existing timer without setting up a new one.

19.6 Miscellaneous Kernel Calls

19.6.1 READ TIME-OF-DAY CLOCK

Trap Name:	T.KERNEL
Action Value (DO.B):	K.READCLOCK

Additional Call Parameters:

none

Return Parameter:

D1.L : time-of-day clock value

Error Returns:

none

This call returns the current value of the one second time-of-day clock. The (unsigned) value is conventionally the number of seconds since 0000 hours on 1st January 1970.

19.6.2 SET TIME-OF-DAY CLOCK

Trap Name:	T.KERNEL
Action Value (DO.B):	K.SETCLOCK

Additional Call Parameters:

D1.L : new value of time-of-day clock

Error Returns:

none

This call sets a new value into the time-of-day clock. The (unsigned) value is conventionally the number of seconds since 0000 hours on 1st January 1970.

Note that changing the time-of-day clock has no effect on any interval timer waits.

19.6.3 CHECK BATTERY

Trap Name:	T.KERNEL
Action Value (DO.B):	K.BATTERY

Additional Call Parameters:

none

entry. If the first byte is zero, then no tone will be generated for that period.

The duration of the tone is given by the second byte as a count of display frame interrupts (i.e. in units of approximately 20 ms). A value of 255 causes the tone to be generated indefinitely. A value of zero causes the buffer pointer and unused length to be reset to the original call values (and hence the first byte of such an entry is ignored).

If the tone sequence terminates voluntarily, and an event number was passed in D3, the specified activity will be notified of the event. If the target activity identifier is passed as -1, the event will be signalled to the calling activity. If no event is required (D3.W = -1) then the target activity identifier is ignored.

Note that only one tone buffer is extant at any given time, and another call will cancel the effect of any previous and still running tone buffer, without causing an event to be notified to the previous caller. Applications should not normally make this call directly, but use the appropriate Director facilities.

Note also that if the first of the two byte entries in the tone buffer contains zero in the second byte, return is made immediately with error BP - bad parameter.

19.6.5 TRACE EXECUTION

Trap Name:	T.TRACE
Action Value (D0.B):	none

Additional Call Parameters:

see below

Error Returns:

none

This call enables an activity to insert an entry into Kernel's execution trace circular buffer. The trace buffer itself is only available in certain development versions of Kernel and must not be used in released code. In other versions this call has no effect. The information stored in the trace entry comprises the 8 bytes immediately following the trap call, (used as identification), and the contents of registers D0, D1, D2, A0, A1 and A4. Return is made eight bytes beyond the call, with all registers and the condition code preserved. The call does not have any scheduling side effects.

It is tentatively suggested that the 8 character identifiers used should follow the project naming standards.

20.7 Keyboard

Following initialisation, the keyboard is in 'system key processing mode', with no dial state set. Until a special keyboard channel is opened, any keyboard characters arriving are lost.

Please purchase Image2PDF on http://www.verypdf.com/ to remove this message.

K.<name>	Description	Reference
XXXVECT	System Exception Redirection	4.2.2
SYSERR	System Error	4.3
CHECKADD	Address Space Query	4.5
NEWACT	Create New Activity	5.10.1
XACT	Destroy Activity	5.10.2
SUICIDE	Destroy This Activity	5.10.3
MURDER	Forcibly Destroy Activity	5.10.4
EXVECT	Notify Activity Exception Vector	5.10.5
GIVACT	Give Activity Status	5.10.6
PRIORITY	Change Activity Priority	5.10.7
GIVSUBSID	Give Activity's Subsidiaries	5.10.8
ASKEVENT	Request Future Event Notification	6.8.1
CLREVENT	Clear Event Request	6.8.2
GLOBEVENT	Signal Global Event(s)	6.8.3
LOCEVENT	Signal Local Event(s)	6.8.4
WAIT	Wait for Event(s) or Specified Period	6.8.5
NEWSEM	Create New Semaphore	7.4.1
XSEM	Destroy Semaphore	7.4.2
SEMOWN	Change Semaphore Ownership	7.4.3
LOCK	Lock Semaphore	7.4.5
LOCKEVENT	Lock Semaphore and Check for Events	7.4.6
UNLOCK	Unlock Semaphore	7.4.7
NEWSEG	Create New Segment	8.5.1
XSEG	Destroy Segment	8.5.2
SEGOWNER	Change Segment Ownership	8.5.3
SEGLIMIT	Set Segment Limites	8.5.4
SEGEXPAND	Expand Segment	8.5.5
SEGREDUCE	Reduce Segment	8.5.6
FREEZE	Freeze Segment	8.5.7
THAW	Thaw Segment	8.5.8
GIVSEG	Give Segment Information	8.5.9
GIVMEMORY	Give Memory Space Information	8.5.10
NEWCELL	Create New Cell	8.5.11
XCELL	Destroy Cell	8.5.12
CELLEXPAND	Expand Cell	8.5.13
CELLREDUCE	Reduce Cell	8.5.14
GIVCFREE	Give Free Cell Space	8.5.15
GIVCLEN	Give Cell length	8.5.16
GIVCSEG	Give Segment Identifier of Cell	8.5.17
XXXCEL	Reset Cell Allocator for Segment	8.5.18
CELL1	Get First Cell of Segment	8.5.19
SQUASH	Squash Cells Down and Free Space	8.5.20

Table A.1 - (a)

Kernel Call Summary

K.<name>	Description	Reference
OPENTIM	Open Interval Timer Channel	18.2.1
TIMER	Set Interval Timer	18.2.2
READCLOCK	Read Time-of-Day Clock	19.6.1
SETCLOCK	Set Time-of-Day Clock	19.6.2
BATTERY	Check Battery	19.6.3
SOUND	Make Sound	19.6.4

Table A.1 - (c)

Kernel Call Summary

Code	Meaning
BL	battery low
BF	block transfer failure
BO	buffer overflow
BP	bad parameter
CF	communications channel failure
CO	catalogue open
CP	CMOS RAM corrupt
DE	data or device error
DI	data invalid
DL	incoming data lost
DT	Director tables full
EX	object already exists
IU	object in use
MD	other microdrive in use
NA	not a valid activity
NB	no immediate byte
NC	operation not complete
ND	not a valid database
NF	object not found
NI	facility not implemented
NL	not a valid semaphore/lock
NO	channel not open
NP	not a valid program
NS	not a valid segment
NT	not a valid telephone line
NV	not a valid volume
OM	out of memory
OP	out of parameters
OR	out of range
OS	out of sectors
PF	input data parity failure
SL	save/load in progress
SR	screen resolution incorrect
TX	transfer cancelled by Kernel
UF	unformatted volume
US	device unserviceable
UV	unexpected event
WF	wrong format
WP	write protected

Table B.1

Error Return Code Summary

Appendix D : List of Logical Device Numbers

The logical device numbers are as follows:

DV.SCREEN	:	the Application Screen
DV.NOTICE	:	the Noticeboard
DV.KEYBD	:	the normal keyboard
DV.SKEYBD	:	the system keyboard
DV.LINE1	:	telephone line 1
DV.LINE2	:	telephone line 2
DV.MODEM	:	the modem
DV.SYNTH	:	the voice synthesiser
DV.FILE	:	microdrive file
DV.PRINT	:	the printer
DV.PIPE	:	pipe channel

Appendix F : Screen Driver Character Sets

F.1 Because the Screen Driver allows the use of multiple and user defined fonts, the character set is not fixed. The character shapes in the default primary font are given in Appendix E (for values in the range $20 to $7F).

F.2 The character shapes in the default secondary font are given in Table F.1 below.

F.3 The format of a normal user-defined character font is a contiguous area of 10 byte entries. Each entry contains a 6 x 10 graphic cell where each byte represents a pixel line, and the least significant 6 bits of each byte represent the pixel columns. The relevant bits are set 'on' for ink colour, and for 'off' paper colour. For example, the font entry for a '!' might be encoded as follows :-

```
         bit   7 6 5 4 3 2 1 0

    byte 0     0 0 0 0 0 0 0 0      ← top guard band
         1     0 0 0 0 0 1 0 0
         2     0 0 0 0 0 1 0 0
         3     0 0 0 0 0 1 0 0
         4     0 0 0 0 0 1 0 0
         5     0 0 0 0 0 1 0 0
         6     0 0 0 0 0 0 0 0
         7     0 0 0 0 0 1 0 0
         8     0 0 0 0 0 1 0 0
         9     0 0 0 0 0 0 0 0

                   ↑
                   left
                   guard
                   band
```

The guard bands are conventionally at the top and left of the cell, but block graphic characters (such as Prestel Mosaics) will occupy the whole 6 x 10 cell, with no guard bands.

F.4 The format of a 64 column mode user-defined character font is a contiguous area of 10 byte entries. Each entry contains a 7 x 10 graphic cell where each byte represents a pixel line, and the least significant 7 bits of each byte represent the pixel columns. The relevant bits are set 'on' for ink colour, and for 'off' paper colour. For example, the font entry for a '!' might be encoded as follows :-

Hex Value	Decimal Value	Character Shape	Notes	Comments
$C0	192	f	1	(function key qualifier)
$C1	193	←		(left arrow)
$C2	194	$^1/_2$		(half fraction)
$C3	195	→		(right arrow)
$C4	196	↑		(up arrow)
$C5	197	↓		(down arrow)
$C6	198	$^1/_4$		(quarter fraction)
$C7	199	‖		(Prestel double bars)
$C8	200	$^3/_4$		(three-quarter fraction)
$C9	201	÷		(divide sign)
$CA	202	#		(hash sign)
$CB	203	©		(single char copyright sign)
$CC	204	_	2	(full width underline)
$CD	205	©	4	} (two char copyright sign)
$CE	206		2,4	}
$CF	207	—	2	(full width mid-line)
$D0	208	∝		(Greek alpha symbol)

Table F.1

Default Secondary Character Font - Additional Shapes

Notes:
1. extends into top guard band, which gets truncated in 8 pixel height characters.
2. extends into left guard band, causing odd results if normal flashing is used.
3. these shapes and encodings are not in accordance with ISO 646 and must be regarded as likely to change in future versions of the OPD.
4. used together, these two characters produce a readable copyright symbol.

Hex Value	Decimal Value	Name	Meaning
$41	65	CLREL	Clear to end of line
$42	66	CLREW	Clear to end of window
$43	67	SVCUR	Save cursor position
$44	68	RSCUR	Restore cursor position
$45	69	PSCR1	Partial Scroll 1
$46	70	PSCR2	Partial Scroll 2
$47	71	PSCR3	Partial Scroll 3
$48	72	PSCR4	Partial Scroll 4
$49	73	PFL	Prestel Flash Left
$4A	74	CLRXX	Clear Whole Screen Area
$4B	75	PFR	Prestel Flash Right
$4C	76	ULINE	Underlining On
$4D	77	XULINE	Underlining Off
$4E	78	XPFL	Special Prestel Flash Left
$4F	79	XPFR	Special Prestel Flash Right

Table F.3

Escape Code Encoding

Appendix G : Keyboard Character Set

A valid combination on the OPD consists of any single key (other than SHIFT, ALT and CONTROL) alone or in combination with one only of the SHIFT, ALT and CONTROL keys; or one of the four cursor control keys together with any combination of the SHIFT, ALT and CONTROL keys. Any other combination is illegal, and is ignored.

Table G.1 lists the keyboard character codes and the key combinations which produce them. Any code that does not have a combination documented cannot be generated from the keyboard. A list of the codes generated for each valid combination of keys is given in Appendix H.

Notes:-

a) the suffixes ".a", ".c" and ".s" etc, denote the key is held down in combination with the ALT, CTRL and SHIFT keys respectively

b) '0' to '9' and '*' refer to the top row of the keyboard 'KO' to 'K9' and 'K*' to the numeric key pad

Hex Value	Decimal Value	Name/ Meaning	OPD Key Combination	Notes
$20	32	space	space	
$21	33	!	1.s	
$22	34	"	'.s	
$23	35	£	3.s	
$24	36	$	4.s	
$25	37	%	5.s	
$26	38	&	7.s	
$27	39	'	'	apostrophe
$28	40	(	9.s	
$29	41	)	0.s	
$2A	42	*	8.s or k*	
$2B	43	+	=.s	
$2C	44	,	,	
$2D	45	-	-	
$2E	46	.	.	
$2F	47	/	/	
$30	48	0	0 or K0	
$31	49	1	1 or K1	
$32	50	2	2 or K2	
$33	51	3	3 or K3	
$34	52	4	4 or K4	
$35	53	5	5 or K5	
$36	54	6	6 or K6	
$37	55	7	7 or K7	
$38	56	8	8 or K8	
$39	57	9	9 or K9	
$3A	58	:	;.s	
$3B	59	;	;	
$3C	60	<	,.s	
$3D	61	=	=	
$3E	62	>	..s	
$3F	63	?	/.s	

Table G.1 - (b)

Hex Value	Decimal Value	Name/ Meaning	OPD Key Combination	Notes
$60	96	`	4.a	grave
$61	97	a	A	
$62	98	b	B	
$63	99	c	C	
$64	100	d	D	
$65	101	e	E	
$66	102	f	F	
$67	103	g	G	
$68	104	h	H	
$69	105	i	I	
$6A	106	j	J	
$6B	107	k	K	
$6C	108	l	L	
$6D	109	m	M	
$6E	110	n	N	
$6F	111	o	O	
$70	112	p	P	
$71	113	q	Q	
$72	114	r	R	
$73	115	s	S	
$74	116	t	T	
$75	117	u	U	
$76	118	v	V	
$77	119	w	W	
$78	120	x	X	
$79	121	y	Y	
$7A	122	z	Z	
$7B	123	{	7.a	
$7C	124	|	1.a	
$7D	125	}	8.a	
$7E	126	tilde	6.a	
$7F	127	del	-.a	'splodge'

Table G.1 - (d)

Hex Value	Decimal Value	Name/ Meaning	OPD Key Combination	Notes
$A0	160			
$A1	161	alt A	A.a	
$A2	162	alt B	B.a	
$A3	163	alt C	C.a	
$A4	164	alt D	D.a	
$A5	165	alt E	E.a	
$A6	166	alt F	F.a	
$A7	167	alt G	G.a	
$A8	168	alt H	H.a	
$A9	169	alt I	I.a	
$AA	170	alt J	J.a	
$AB	171	alt K	K.a	
$AC	172	alt L	L.a	
$AD	173	alt M	M.a	
$AE	174	alt N	N.a	
$AF	175	alt O	O.a	
$B0	176	alt P	P.a	
$B1	177	alt Q	Q.a	
$B2	178	alt R	R.a	
$B3	179	alt S	S.a	
$B4	180	alt T	T.a	
$B5	181	alt U	U.a	
$B6	182	alt V	V.a	
$B7	183	alt W	W.a	
$B8	184	alt X	X.a	
$B9	185	alt Y	Y.a	
$BA	186	alt Z	Z.a	
$BB	187		k5.c or 5.c	
$BC	188		k6.c or 6.c	
$BD	189		k7.c or 7.c	
$BE	190		k8.c or 8.c	
$BF	191		k9.c or 9.c	

Table G.1 - (f)

Hex Value	Decimal Value	Name/ Meaning	OPD Key Combination	Notes
$E0	224	recall	recall	4
$E1	225	spkr	spkr	4
$E2	226	redial	redial	4
$E3	227	hold+s	k1.s	4
$E4	228	select	k2.s	4
$E5	229	end	k3.s	4
$E6	230	dial	k4.s	4
$E7	231	hold	k5.s	4
$E8	232	time	k8.s	4
$E9	233			
$EA	234			
$EB	235			
$EC	236			
$ED	237			
$EE	238			
$EF	239			
$F0	240	f0	k0.a	
$F1	241	f1	k1.a	
$F2	242	f2	k2.a	
$F3	243	f3	k3.a	
$F4	244	f4	k4.a	
$F5	245	f5	k5.a	
$F6	246	f6	k6.a	
$F7	247	f7	k7.a	
$F8	248	f8	k8.a	
$F9	249	f9	k9.a	
$FA	250			
$FB	251			
$FC	252			
$FD	253			
$FE	254			
$FF	255			

Table G.1 - (h)

Note 4 : these codes only appear as keyboard input on telephone channels.

LEGEND	No shifts		SHIFT		ALT	CTL	
A	61	a	41	A	A1	01	soh
B	62	b	42	B	A2	02	stx
C	63	c	43	C	A3	03	etx
D	64	d	44	D	A4	04	eot
E	65	e	45	E	A5	05	enq
F	66	f	46	F	A6	06	ack
G	67	g	47	G	A7	07	bel
H	68	h	48	H	A8	08	bs
I	69	i	49	I	A9	09	ht
J	6A	j	4A	J	AA	0A	lf
K	6B	k	4B	K	AB	0B	vt
L	6C	l	4C	L	AC	0C	ff
M	6D	m	4D	M	AD	0D	cr
N	6E	n	4E	N	AE	0E	so
O	6F	o	4F	O	AF	0F	si
P	70	p	50	P	B0	10	dlc
Q	71	q	51	Q	B1	11	dc1
R	72	r	52	R	B2	12	dc2
S	73	s	53	S	B3	13	dc3
T	74	t	54	T	B4	14	dc4
U	75	u	55	U	B5	15	nak
V	76	v	56	V	B6	16	syn
W	77	w	57	W	B7	17	etb
X	78	x	58	X	B8	18	can
Y	79	y	59	Y	B9	19	em
Z	7A	z	5A	Z	BA	1A	sub

LEGEND	no shifts	SHIFT	ALT	CTL
START	D0 start	D9 blank	////	DA reset
RESUME	D1 resume	D1 resume	////	////
REVIEW	D2 review	D2 review	////	////
TAB	09 tab	C5 back-tab	////	C6 format
INS DEL	C2 delete	C3 insert	C9	C4 remove
ENTER RET	0D cr	0A lf	C7	////
(space)	20 space	20 space	20 space	C8 break-in
RECALL	E0 recall	D3 list	////	////
SPKR	E1 spkr	DF auto	////	////
REDIAL	E2 redial	D4 last	////	////
H/S 1	31 1	E3 h/s	F1 f1	1C fs
SEL 2	32 2	E4 sel	F2 f2	1D gs
END 3	33 3	E5 end	F3 f3	1E rs
DIAL 4	34 4	E6 dial	F4 f4	1F us
HOLD 5	35 5	E7 hold	F5 f5	BB
SHOW 6	36 6	D6 show	F6 f6	BC
CAPS 7	37 7	C0/1 caps	F7 f7	BD
TIME 8	38 8	E8 time	F8 f8	BE
LOOK 9	39 9	D7 look	F9 f9	BF
0	30 0	////	F0 f10	00 nul
ESC *	2A *	1B esc	////	////
PRNT #	CA #	D8 print	DB printa	CF printm

Appendix I : Telephone Driver Character Codes

The encodings of the output control bytes that may be written to a telephone channel are shown in Table I.1. For completeness, this table includes some values used internally by Kernel, that are not documented in the body of this specification.

The encodings of the status information bytes that may be read from a telephone channel are shown in Table I.2.

The type of recall mechanism for a given telephone line is governed by non-volatile memory (CMOS) entries as follows:-

entry \$20 - line 1 configuration
entry \$21 - line 2 configuration

If the relevant entry for a line is not present, then a normal Earthed Recall is used. If the relevant entry is present, then a Timed Break Recall is used, and the entry must have the following format:

i	b	f

i : initial mute time (1 to 255)
b : line break time (1 to 255)
f : final mute time (1 to 255)

All the times are in units of 20 ms, and the initial mute time will in practise vary from 20(i-1) ms to 20i ms. If any of the fields is zero, the effect is undefined.

Note that the values in these entries (or even the existence of the entries) should not normally be altered, except by a configuration program valid in the territory or area concerned.

Hex Value	Decimal Value	Name	Meaning
$20	32	DTMF0	Send DTMF data $00
$21	33	DTMF1	Send DTMF data $01
$22	34	DTMF2	Send DTMF data $02
$23	35	DTMF3	Send DTMF data $03
$24	36	DTMF4	Send DTMF data $04
$25	37	DTMF5	Send DTMF data $05
$26	38	DTMF6	Send DTMF data S06
$27	39	DTMF7	Send DTMF data $07
$28	40	DTMF8	Send DTMF data $08
$29	41	DTMF9	Send DTMF data $09
$2A	42	DTMFA	Send DTMF data $0A
$2B	43	DTMFB	Send DTMF data $0B
$2C	44		
$2D	45		
$2E	46		
$2F	47		
$30	48	DIAL0	Dial digit 0
$31	49	DIAL1	Dial digit 1
$32	50	DIAL2	Dial digit 2
$33	51	DIAL3	Dial digit 3
$34	52	DIAL4	Dial digit 4
$35	53	DIAL5	Dial digit 5
$36	54	DIAL6	Dial digit 6
$37	55	DIAL7	Dial digit 7
$38	56	DIAL8	Dial digit 8
$39	57	DIAL9	Dial digit 9
$3A	58	DIAL*	Dial *
$3B	59	DIAL#	Dial #
$3C	60		
$3D	61		
$3E	62		
$3F	63		

Table I.1 - (b)

Phone Driver Output Codes

Appendix J : Microdrive Volume Formats

J.1 Volume Description Record

The Volume Description Record is held in the first block of each catalogue file, and contains three main units of information, as follows:-

Volume Statistics
Volume Name
Sector Allocation Map

The format of the volume description record is given in Table J.1.

Notes:

a) the TIME field is not cleared to zero if a volume is re-formatted.

b) the transfer and fail counts are not cleared to zero if a volume is re-formatted.

c) the WFP (see byte 20 in Table J.1) count does not include any that occur during the format operation.

d) following the initial format of a virgin cartridge the difference between the FLAWS field and the WFP field give the number of flaw sectors following the format process.

e) for entries with sector numbers greater than or equal to the size field in the Volume Statistics, the relevant sector entry is undefined.

f) if the sector allocation table is read from the volume, then a non-zero WOPEN field means that the table is corrupt or invalid and must not be used.

g) if the ICL field is not 'ICL ' then the volume is assumed to be in some other format, and not used.

h) the 'volume is write protected' flag (bit 7 of the FLAGS field) only appears in the store held version of the Volume Description Record, and not on the cartridge itself.

i) the 'volume not formatted by Kernel' flag (bit 0 of the FLAGS field) must be set by any program that does not use Kernel to format the cartridge and create the catalogue (for example a sector by sector copy utility).

j) the Volume Uniqueness Bytes are set to a unique 4 byte value when the volume is formatted. The value is not preserved by any sector by sector cartridge copying mechanism supplied by ICL.

description entry returned from a GIVE FILE INFORMATION IMMEDIATE call.

Sector Allocation Map Area:

38	WOPEN	count of files open for writing
39	FREE	count of free sectors
40-489	SAM	sequence of two byte fields where (40 + 2n) is the entry for sector n
40+2n		0 : sector is free $FF : sector is flawed 0<m<255 : in use by file number 'm'
41+2n		block number within file number 'm' (undefined if sector is free or flawed)

Volume Uniqueness Bytes:

508-511	UNIQ	unique four byte value

Table J.1
Volume Description Record Format - (ii)