SYSCALLS.1

PSIONICS FILE - SYSCALLS.1

==========

System calls (part 1)

Last modified 1997-09-09

==========

The Psion System 3 operating system offers a large number of system calls.
A system call is identified by either a "function number" or by a "functionnumber and a "subfunction number". These are abbreviated to "Fn" and "Sub"
in the descriptions of system calls.
Each call can take up to 5 word parameters, called BX, CX, DX, SI, and DI.In addition, there may be up to two byte parameters (AH and AL) or anadditional word parameter (AX). The values returned from the call have thesame structure; in addition, there is an error flag and three result flags
(the latter are only used by a few system calls). Note that AX is equivalentto (AH * 256 + AL); the description will use whichever is most convenient.
There are two OPL keywords for making system calls: CALL and OS. They accessthe same system calls, and differ only in the way in which the parameters are
passed and the results returned.
The CALL keyword takes from 1 to 6 arguments. If an argument is omitted, thecorresponding parameter is set to an undefined value. The first argument
should be one of:

    Fn
    Fn + 256 * Sub
    Fn + 256 * AH

according to the specific system call. The remaining arguments provide the
word parameters:

    argument 2: BX
    argument 3: CX
    argument 4: DX
    argument 5: SI
    argument 6: DI

The keyword returns a value, which is one of the results:

```
    AX
    AH * 256 + AL
(which are equivalent). The flags and the other 5 results are not available.
OS takes two or three arguments; if there is no third, it is taken to beidentical to the second.
The first argument is Fn, while the second and thirdare each the address of a 12 byte buffer,
holding the parameters and results
respectively:
  Offset  0 (word): one of:
    Sub * 256 + AL    (parameters only)
    AX                (parameters and results)
    AH * 256 + AL     (parameters and results)
  Offset  2 (word): BX
  Offset  4 (word): CX
  Offset  6 (word): DX
  Offset  8 (word): SI
  Offset 10 (word): DIThe keyword returns a value which represents the error or result flags.
Each flag is held in a specific bit of the value; the remaining bits are
unspecified:
    Bit  0: UL flag or error flag (depending on context)
    Bit  6: EQ flag
    Bit  7: SL flag
Thus, if the call can fail, it has failed if the returned value is odd.
When a parameter or result is described as a cstr, a qstr, a buffer, or anyother construct
requiring more than 2 bytes, the actual parameter or result is
the address of the construct.
When a register holds an address (for the above reason or otherwise), it isrelative to some
segment register. If programming entirely in OPL, all therelevant segment registers (DS, SS, and
ES) are the same. However, if assembleris being used, it is sometimes useful to have ES
different from the others(see the Psionics file PROCESS for more details). A prefix of "e" to a
register
name (such as "eBX") means that the address is relative to ES. Unprefixed namesare relative to
DS or SS (which should be the same). A prefix of "u" means
that the segment register is unknown.
The system calls are described in a standard form. The first line gives thefollowing information:
* the Fn value
* the Sub value if any
The second line gives the following:
```

* The name given to the call by the SDK.* A note such as "v3.1" giving the first version of the operating system
  to support the call; if omitted, all versions support it.* The word "fails" if the call can fail; if omitted, the call never fails. - If a call can fail, then the error flag shows whether or not it did so. If   it failed, AL holds an error code between 128 and 255; this is normally
    treated as a negative number between -1 and -128, found by evaluating   the expression (AX OR $FF00). AH, and any of the 5 word results that the   call would have changed, are unspecified.
  - If a call cannot fail, the error flag may still be set; this should be
    ignored.
  - Even if a call cannot fail, bad parameters can cause a system panic.* The word "async" if the call is always asynchronous, or the word "vsync" if it is sometimes asynchronous; if omitted, the call is synchronous. - A synchronous call is "completed" (has carried out all its actions) when
    the system call returns. - An asynchronous call returns immediately, even though the call might not
    have completed. A parameter of the call specifies a "status word", which   the call initially sets to -46 (the error code meaning "not completed).   When the relevant action has been taken or event occurs, the call   is completed, the status word is changed to indicate the result of the
    call, and an IOSIGNAL is sent to the process. [This is exactly the same   behaviour as the IOA keyword.]
  - If a call is "vsync", then it may be made as either a synchronous or   asynchronous call:
    + to make a synchronous call, set AL to zero (the parameter giving the   address of the status word will be ignored);   + to make an asynchronous call, set AL to any other value (the address
      of the status word must be valid);   + descriptions of such calls will omit AL from the list of parameters.
  - An asynchronous call may complete before returning (for example if the   relevant event has already happened). In this case the process will never   see the status word with the value -46, but an IOSIGNAL is still received. - Some asynchronous calls have a corresponding "cancel" call. This causes
    the original call to be completed with the status word set to some   negative value other than -46.
This is then followed by a list of the parameters and results used by thecall; any parameter not

```
listed is ignored. For the 5 main word results, theresult (if the call does not fail) is the
same as the corresponding parameter(including when the former is ignored) unless a different
meaning is shownfollowing an arrow (->). For the results AH, AL, and AX, and also the result
flags, the result is unspecified if no such description is given.
For example, consider the following (non-existent) system call:
Fn $12 Sub $34
GetMagicCookie fails
    AX: -> magic cookie table entry zero
    BX: table entry number
    CX: -> magic cookie entry
    SI: addend -> CX out + SI in
Returns entries from the magic cookie table.
This system call could be used either by:
    entry0% = CALL ($3412, entry%, 0, 0, addend%)which only returns magic cookie table entry
zero, or by:
    LOCAL regs%(6)
    regs%(1)=$3400
    regs%(2)=entryno%
    regs%(5)=addend%
    failed%=1 AND OS ($12, ADDR(regs%()))
    REM regs%(2,4,6) have not changed
    IF NOT failed
        entry0%=regs%(1)
        entry%=regs%(3)
        REM regs%(5) will equal entry%+addend%
    ELSE
        REM regs%(1,3,5) are undefined
        error%=regs%(1) and $FF00
    ENDIF
which returns much more information.
System calls
------------
Note that some system calls are deliberately not described.
Fn $80 Sub $00
SegFreeMemory
    AX: -> free memory
Returns the amount of free system memory, in units of 16 bytes.
```

```
Fn $80 Sub $01
SegCreate fails
    AL: segment type: ordinary processes should use 1.
    AX: -> segment handle
   eBX: (cstr) name of the segment
    CX: segment size in units of 16 bytesCreates an additional memory segment. Each segment must be given a name, of theform "8.3" (i.e. up to 8 characters, optionally followed by a dot and up to 3
more characters). If the name is already in use, the call will fail. Oncecreated, the segment may be used by the process as additional memory, eithervia SegCopyFrom and SegCopyTo, or in assembler code (see the Psionics file
KERNEL).
Fn $80 Sub $02
SegDelete fails
   eBX: segment handleDeletes an additional memory segment. If any other process has opened the segment, the call will fail.
Fn $80 Sub $03
SegOpen fails
    AX: -> segment handle
   eBX: (cstr) name of the segmentOpens the additional memory segment with the given name (if no such segment
exists, the call will fail). The call will fail if the process has the segmentopen already. Except where stated, all calls using additional memory segmentsmust be given handles from SegCreate or SegOpen calls.
Fn $80 Sub $04
SegClose fails
    BX: segment handleCloses an additional memory segment which the process has open. If this was the
only process with the segment open, it is deleted. Open segments are closedwhen a process terminates.
Fn $80 Sub $05
SegSize
    AX: -> segment size
    BX: segment handle
Returns the size of an additional memory segment in units of 16 bytes.
Fn $80 Sub $06
SegAdjustSize fails
```

BX: segment handle

CX: new size in units of 6 bytesChanges the size of an additional memory segment. The memory will be added to
or removed from the end of the segment.

Fn $80 Sub $07

SegFind fails

AX: -> segment handle

BX: last segment handle

SI: address of 14 byte buffer used by the call

eDI: (cstr) pattern to search forFinds the additional memory segments with names matching the search pattern
(use ? and * as wild cards). The first call should be made with the lastsegment handle set to 0, and subsequent calls should set it to the handlereturned by the previous call. The call fails when there are no more segmentsto return. It is not necessary to keep calling SegFind until this happens.

Fn $80 Sub $08

SegCopyTo

BX: segment handle

CX: number of bytes to copy

DX: high half of start address in segment

SI: address in process of first byte to copy DI: low half of start address in segment

Copies data from the current process to an additional memory segment.

Fn $80 Sub $09

SegCopyFrom

BX: segment handle

CX: number of bytes to copy

DX: high half of start address in segment

SI: address in process to copy first byte to DI: low half of start address in segment

Copies data from an additional memory segment to the current process.

Fn $80 Sub $0A

SegLock

BX: segment handleLocks an additional memory segment; a locked segment will not be deleted even
if no processes have it open.

Fn $80 Sub $0B

SegUnLock

BX: segment handleUnlocks an additional memory segment. The number of unlock calls should

equal
the number of lock calls; additional unlocks may cause the segment to be
deleted while still open.
Fn $80 Sub $0C
SegRamDiskUsed
    AX: -> size of the ram disc in units of 16 bytesReturns the size of the ram disc; this should be treated carefully, as it
can change without warning.
Fn $80 Sub $0D
SegCloseLockedOrDevice
    BX: segment handleUnlocks an additional memory segment where the handle may be that for a different process. See SegUnLock for more details.
Fn $81 Sub $00
HeapAllocateCell fails
    AX: -> allocated block
    CX: number of bytes to allocateAllocates a block of memory from the heap. The returned block may be larger
than requested. Heap memory is part of the process's normal memory segment.
Fn $81 Sub $01
HeapReAllocateCell fails
    AX: -> new address of block
    BX: block to be reallocated, or zero
    CX: new size of block in bytesChanges the size of a heap block; the block may have to be moved to do this.The memory will be added or deleted to the end of the block. If the call fails, the block is unaffected. If BX is zero, then a new cell is allocated. The newblock may be larger than requested.
Fn $81 Sub $02
HeapAdjustCellSize fails
    AX: -> new address of block
    BX: block to be adjusted (must not be zero)
    CX: number of bytes to add or remove
    DX: location within block of the changeChanges the size of a heap block; the block may have to be moved to do this.If the call fails, the block is unaffected. The memory will be added or removed
at the offset given by DX. For example, if the block was 10 bytes long, CX is3, and DX is 6, the new block will be 13 bytes, and consist of the first 6bytes of the old block, then 3 random bytes, then the last 4 bytes of the old

block. The new block may be larger than requested.

Fn $81 Sub $03

HeapFreeCell

    BX: block to free

Frees a previously allocated heap block.

Fn $81 Sub $04

HeapCellSize

    AX: -> cell size in bytes

    BX: address of blockReturns the actual size of a heap block (which may be larger than requested

when the block was created or last resized).

Fn $81 Sub $05

HeapSetGranularity

    BX: new granularity in units of 16 bytesWhen the application data space is not large enough to satisfy a heap request,it is grown by the granularity. The default is 2kb (128 units), and the maximum is 16k (1024 units).

Fn $81 Sub $06

HeapFreeMemory

    AX: -> maximum possible free space on heap

    BX: -> address of start of heapReturns the amount of heap space which can be allocated (by using freed blocksor by growing the data space, or both), plus the address of the base of the heap.

Fn $82 Sub $00

SemCreate fails

    AX: -> semaphore handle

    BX: initial value of semaphoreCreates a new semaphore for process interlocking. Each semaphore has a list ofprocesses associated with it (this list is initially empty) and a value, which

must initially be zero or positive.

Fn $82 Sub $01

SemDelete

    BX: semaphore handleDeletes a semaphore. Any processes on the semaphore's list will be restarted.

When a process terminates, any semaphore it created will be deleted.

Fn $82 Sub $02

SemWait

    BX: semaphore handleIf the value of the semaphore is positive or zero, one is subtracted

from it.
If the value is now or initially -1, the current process is then blocked andadded to the end of the list for that semaphore.
Fn $82 Sub $03
SemSignalOnce
    BX: semaphore handleIf the list for the semaphore is not empty, the first process on the list is
removed and restarted. If the list is empty (either initially or after theonly process is removed), one is added to the value of the semaphore. If therestarted process has a higher priority than the current one, a reschedulewill take place and the other process will start running (see the description
of priorities in the Psionics file PROCESS).
Fn $82 Sub $04
SemSignalMany
    BX: semaphore handle
    CX: number of calls to make (must be positive)This is equivalent to making several calls to SemSignalOnce.
Fn $82 Sub $05
SemSignalOnceNoReSched
    BX: semaphore handleThis is identical to SemSignalOnce except that a reschedule never takes place
because of the call (though one may take place due to the current process using
up its time slot).
Fn $83 Sub $00
MessInit fails
    BX: number of message slots + 256 * maximum data lengthInitialize the message system so that the current process can receiveinter-process messages. The call reserves the indicated number of message
slots, with each slot having room for the indicated amount of data. The slotsare allocated on the heap.
Fn $83 Sub $01
MessReceiveAsynchronous async
    BX: address of message slot pointer
    DI: address of the status wordWhen a message arrives, the message slot pointer is set to the address of the
message, the status word is set to zero, and the call completes. The message
has the format:

```
  Offset  0 to   3: used by the message system
  Offset  4 (word): message type
  Offset  6 (word): process sending the message
  Offset  8 onward: data in message
Fn $83 Sub $02
MessReceiveWithWait
    BX: address of message slot pointerWhen a message arrives (this may have happened before the
call; otherwise thecall waits until a message arrives), the message slot pointer is set to the
address of the message. The message format is as for MessReceiveAsynchronous.
Fn $83 Sub $03
MessReceiveCancel
Cancel any pending MessReceiveAsynchronous.
Fn $83 Sub $04
MessSend fails
    BX: process to receive the message
    CX: message type
    SI: address of first byte of message dataSend a message to a process. The call will block
until there is a free messageslot in the receiving process. The receiving process determines the
amount of
data sent. The reply is ignored.
If the sender has priority 128 or above, the message is placed at the front ofthe queue of
messages waiting for the recipient (if it is not already waitingfor a message). Otherwise it is
placed at the back of the queue.
Fn $83 Sub $05
MessSendReceiveAsynchronous fails async
    BX: process to receive the message
    CX: message type
    SI: address of first byte of message data
    DI: address of the status wordSend a message to a process; when the recipient replies, the
status word isset to the reply and the process is sent an IOSIGNAL. The call will block until
there is a free message slot in the receiving process.
Fn $83 Sub $06
MessSendReceiveWithWait fails
    AX: -> reply
    BX: process to receive the message
    CX: message type
    SI: address of first byte of message dataSend a message to a process, and blocks until the
```

recipient replies.
Fn $83 Sub $07
MessFree
    BX: address of received message
    CX: reply
The reply is sent to the recipient of the message, and the message slot isfreed and can be used for another incoming message.
Fn $83 Sub $08
MessSignal fails
    BX: process to watch
    CX: message typeRequests that, when the specified process terminates, the kernel sends a message of the indicated type to the current process. The data in the message
has the format:
  Offset 0 (word): process id of terminating process Offset 2 (byte): reason code (see ProcKill) or panic code
  Offset 3 (byte):
    0 = process terminated or was killed
    1 = process caused a panic
    2 = a subtask of the process caused a panic in the processIf the message slots do not have room for 4 bytes of data, not all this
information is available.
Fn $83 Sub $09
MessSignalCancel fails
Fn $83 Sub $0A
MessSignalCancelX fails
    BX: process to watch
    CX: message type (MessSignalCancelX only)Cancels a call to MessSignal for the indicated process. MessSignalCancelignores the message type and should not be used if more than one MessSignal
call has been made for the process.
Fn $84 is used to control dynamic libraries and is not described here.
Fn $85 Sub $00
IoOpen fails
    AX: -> handle of opened device
   eBX: (cstr) name of device driver
    CX: mode
    DX: handle of device being attached toOpens a channel to a device driver. If the driver is a

base driver, then
DX is ignored (and this call is equivalent to the IOOPEN keyword). If it isa stacked driver (see the Psionics file DEVICES), then the handle to bestacked on must be specified. The meaning of the mode is determined by thedevice. A driver can support several channels at once, and each has its own
handle. The driver must be a logical device driver.
Fn $85 Sub $01 and $02 involve physical device drivers and should only becalled from within logical device drivers.
Fn $85 Sub $03 to $05 should only be used by the operating system.
Fn $85 Sub $06
DevLoadLDD fails
Fn $85 Sub $07
DevLoadPDD fails
   eBX: (cstr) name of file holding the driverLoads a device driver into the system. A driver cannot be opened until it is
loaded. The correct call for the driver type must be used.
Fn $85 Sub $08
DevDelete fails
   eBX: (cstr) name of the device driver   DX: $DD01 for logical device drivers, or $DD21 for physical device driversUnloads a device driver from the system. Open drivers and those in the ROM cannot be unloaded.
Fn $85 Sub $09
DevQueryUnits fails
    AX: -> number of channels supported   eBX: (cstr) name of the device driver, without a trailing colon
Returns the number of simultaneous open channels supported by a driver (whichmust be a logical one); $FFFF indicates no limit.
Fn $85 Sub $0A
DevFind fails
    AX: -> find code
    BX: last find code   DX: $DD01 for logical device drivers, or $DD21 for physical device drivers
    SI: address of 14 byte buffer
   eDI: (cstr) pattern to search forFinds all devices with names matching the search pattern (use ? and * as wildcards). The first call should be made with the last find code set to 0, and subsequent calls should set it to the code returned by the previous call. Thecall fails when there are no more drivers to return. It is not necessary tokeep calling DevFind until this

```
happens. The names do not have a trailing
colon.
Fn $85 Sub $0B should only be used by the operating system.
Fn $85 Sub $0C is used for special operations on device drivers.
Fn $86 Sub $00
IoAsynchronous fails async
Fn $86 Sub $01
IoAsynchronousNoError async
Fn $86 Sub $02
IoWithWait fails
    AL: service number
    AX: -> result from driver
    BX: handle of driver
    CX: address of first argument
    DX: address of second argument    DI: address of the status word (ignored by IoWithWait)
These calls are equivalent to the IOA, IOC, and IOW keywords respectively[For those without
documentation of IOC, this is the same as IOA, except thaterrors are handled by setting the
status word and calling IOSIGNAL, so that
IOC always succeeds, unlike IOA.]
Fn $86 Sub $03 and $04 should only be used by device drivers.
Fn $86 Sub $05
IoWaitForSignal
This call is equivalent to the IOWAIT keyword.
Fn $86 Sub $06
IoWaitForStatus
    DI: address of the status word
This call is equivalent to the IOWAIT keyword.
Fn $86 Sub $07
IoYield
    DI: address of the status word
This call is equivalent to the IOYIELD keyword.
Fn $86 Sub $08
IoSignal
Sends an IOSIGNAL to the current process. This call is equivalent to the
IOSIGNAL keyword.
Fn $86 Sub $09
IoSignalByPid fails
```

```
Fn $86 Sub $0A
IoSignalByPidNoReSched fails
    BX: process IDSends an IOSIGNAL to a process. With the latter call a reschedule never takes place because of the call (though one may take place due to the currentprocess using up its time slot).
Fn $86 Sub $0B to $0F should only be used by device drivers.
Fn $86 Sub $10
IoClose fails
    AX: -> result from driver
    BX: handle
This call is equivalent to the IOCLOSE keyword.
Fn $86 Sub $11
IoRead fails
Fn $86 Sub $12
IoWrite fails
    AX: -> amount actually read or result from driver
    BX: handle
    CX: address of buffer
    DX: number of bytes read or writtenThese calls are equivalent to the IOREAD and IOWRITE keywords.
Fn $86 Sub $13
IoSeek fails
    AX: -> result from driver
    BX: handle
    CX: mode
    DX: address of long holding seek position
This call is equivalent to the IOSEEK keyword.
Fn $86 Sub $14 should only be used by the window manager.
Fn $86 Sub $15 to $17 should only by used from assembler.
Fn $86 Sub $18
IoShiftStates
    AL: modifiers
This call makes available the state of the various modifier keys:
  Bit 1: shift
  Bit 2: control
  Bit 3: psion
Fn $86 Sub $19
```

IoWaitForSignalNoHandlerThis call should be used instead of IOSEEK by tasks (subsidiary processes of
a process).
Fn $86 Sub $1A
IoSignalKillAsynchronous fails async
    BX: process to watch
    DI: address of the status wordThis call completes, and the status word is set to 0, when the specified
process terminates.
Fn $86 Sub $1B
IoSignalKillCancel fails
    BX: process to watch
Cancel any pending IoSignalKillAsynchronous.
Fn $86 Sub $1C and $1D should only be used by the window manager.
Fn $86 Sub $1E
IoPlaySoundW v3
    AL: -> failure code
    BX: (cstr) sound file name
    CX: duration to play (in 1/32 second)
    DX: volume: 0 (loudest) to 5 (softest)Plays a sound file. A duration of 0 means the file header specifies the
duration; otherwise, the sound is clipped or padded with space as needed.The name may be either an ordinary filename, or a string of the form "*ABCD",
meaning the first found of the files:
    ROM::ABCD.WVE
    LOC::M:\WVE\ABCD.WVE
    LOC::A:\WVE\ABCD.WVE
    LOC::B:\WVE\ABCD.WVENote that this call does not fail in the usual way, but just returns a failure
code.
Fn $86 Sub $1F
IoPlaySoundA v3 async
    BX: (cstr) sound file name
    CX: duration to play (in 1/32 second)
    DX: volume: 0 (loudest) to 5 (softest)
    DI: address of the status wordPlays a sound file asynchronously; the call completes when the sound has

finished. The other arguments are as for IoPlaySoundW.

Fn $86 Sub $20

IoPlaySoundCancel v3

Cancel any pending IoPlaySoundA.

Fn $86 Sub $21

IoRecordSoundW v3

AL: -> failure code

BX: (cstr) sound file name CX: number of samples to record, in units of 2048 samples Records a sound file. Note that 2048 samples are slightly more than a quarterof a second. The file will be created before recording, and must not be ona flash device. Note that this call does not fail in the usual way, but just

returns a failure code.

Fn $86 Sub $22

IoRecordSoundA v3 async

BX: (cstr) sound file name

CX: number of samples to record, in units of 2048 samples DI: address of the status word Records a sound file asynchronously; the call completes when the recording hasfinished. The other arguments are as for IoRecordSoundW.

Fn $86 Sub $23

IoRecordSoundCancel v3

Cancel any pending IoRecordSoundA.

Fn $86 Sub $24

IoPlaySoundAO v3.9 async

BX: (cstr) sound file name

CX: duration to play (in 1/32 second)

DX: volume: 0 (loudest) to 5 (softest)

DI: address of the status word

SI: duration to skip (in 1/32 second)Plays part of a sound file asynchronously, skipping some initial portion ofthe sound; the call completes when the sound has finished. The other arguments are as for IoPlaySoundA.

Fn $87 Sub $00 is carried out automatically for OPL programs.

Fn $87 Sub $01

FilExecute fails vsync

eBX: (cstr) program file name

CX: (qstr) program information

DX: address of the status word DI: address of word to be filled with process ID of new process

Starts a new process and places it in "suspended" state. The program filename will have the extension ".IMG" added if one is not specified. The programinformation is a qstr whose interpretation depends on the program; it isavailable from location 36 of the new process's data segment (see Psionics filePROCESS). The process name will be set to the name part of the file (excluding
any extension). If another process of this name is already running, both mustexecute the same code, or the call will fail.
@@ More in S3/S3a manual chapters 1 and 2.@@ Program information often consists of the following catenated
together (note that all these elements except the first are cstrs):(1) the character 'C' (create new file) or 'O' (open existing file);(2) (cstr) the application name, with the first character uppercased and the
    rest lowercased;
(3) (cstr) one of:
    (a) the normal extension for filenames, including the leading dot;   (b) the normal extension with dot, followed by a space and aliasing
        information;
    (c) an empty string
(4) (cstr) the data file path passed as the initial argument;(5) (cstr) the application path (may be omitted if not required).
A binary application may be used to operate on a standard file type. Examples
include:

Program	Application	Extension	[item 2]	[item 3]
"ROM::DATA.APP"	"Data"	".DBF"	"ROM::WORD.APP"	"Word" ".WRD"
"ROM::WORD.APP"	"Program"	".OPL"		
"LOC::C:\APP\COMMS.APP"	"Comms"	".SCO"		

In each of these cases there is no aliasing information or application path.An example program information qstr is "OData~.DBF~LOC::M:\DBF\MYDATA.DBF~"
(tilde indicates a zero byte).
@This section needs rewriting@The word processor may also be used as a program editor, by making item 2
its name ("Program") and item 3 an extension (usually ".OPL") followed by thenecessary aliasing information: this is empty for normal Word, or:* "O" for OPL editor, "S" for SCR editor, "$" for plain text editor (3a only)
  "/" for custom template (3a only); any letter uses translator SYS$PRG<letter>.IMG to

translate, then
* then "R" indicates that the translator can execute resulting code, anything else means it can't, then
* extension and directory ("OPO" or "SCO" for example) of translated files,* then on 3a only optional "*" to indicate compatibility mode exists.An example program information qstr is "Program~.OPL OROPO~LOC::M:\OPL\MYPROG.OPL~"
template mode for Word means there must be a \WDR\<template>.WRT file on thecurrent drive; template name is that if the application name. For example,
alias file of:

    Letter.LET
    \LET\
    1083
    Word
    /

uses template LETTER. Note mode 80 in application type in alias file.
Finally, a translator may be used to execute a translated file: in the caseof the translator "ROM::SYS$PRGO", it can execute either OPO or OPA files.The application name is given in the APP ... ENDA clause for OPA files, andis "RunOpl" for OPO files; there is no extension or aliasing information.
For OPA files, the data file is passed to the application, and the applicationpath shows which application is actually run. For OPO files, the data fileshould name the OPO file itself, and there is no application path. Examplesare "RunOpl~~M:\OPO\MYPROG.OPO~" and "MyApp~~M:\DAT\MYDATA~M:\APP\MYAPP.OPA~".
Fn $87 Sub $02
FilParse fails vsync

    BX: (cstr) file name to be parsed
    CX: (cstr) file specification to be used
    DX: address of the status word
    SI: address of a 6 byte buffer to be filled in   DI: address of 128 byte buffer to be filled with the parsed filenameThis call is equivalent to the PARSE$ keyword. The buffer is filled as follows:

  Offset 0 (byte): length of node name
  Offset 1 (byte): length of device name
  Offset 2 (byte): length of path name
  Offset 3 (byte): length of base name
  Offset 4 (byte): length of extension
  Offset 5 (byte): flags   Bit 0: the base name or extension contains a wildcard ("*" or

"?")

    Bit 1: the base name contains a wildcard

    Bit 2: the extension contains a wildcard

Fn $87 Sub $03

FilPathGet fails vsync

    BX: 128 byte buffer

    DX: address of the status wordThe buffer is filled with the current filing system default path (a cstr).

Fn $87 Sub $04

FilPathSet fails vsync

    BX: (cstr) new path

    DX: address of the status word

Sets the default path (equivalent to the SETPATH keyword).

Fn $87 Sub $05

FilPathTest fails vsync

    BX: (cstr) pathname to be tested

    DX: address of the status wordEquivalent to the EXIST keyword; the call succeeds if the file with that

pathname exists.

Fn $87 Sub $06

FilDelete fails vsync

    BX: (cstr) pathname to be deleted

    DX: address of the status wordEquivalent to the DELETE keyword; non-empty directories cannot be deleted.

Fn $87 Sub $07

FilRename fails vsync

    BX: (cstr) old pathname

    CX: (cstr) new pathname

    DX: address of the status word

Equivalent to the RENAME keyword.

Fn $87 Sub $08

FilStatusGet fails async

    BX: (cstr) filename

    CX: 16 byte buffer

    DX: address of the status word

The buffer is filled in with information about the file: Offset 0 (word): buffer format version: always 2

```
  Offset  2 (word): attributes
  Offset  4 (long): file size in bytes
  Offset  8 (long): time last modified
The attribute bits have the following meanings (question marks indicate thatthe bit name was
specified but the meaning is unknown; equals signs mean thatthe bit can be set by
FilStatusSet):
  = Bit  0: file is not ReadOnly
  = Bit  1: file is Hidden
  = Bit  2: file is a System file
    Bit  3: file is a Volume name
    Bit  4: file is a Directory
  = Bit  5: file is Modified
  ? Bit  8: file is a readable file
  ? Bit  9: file is an executable file
  ? Bit 10: file is a byte stream file
    Bit 11: file is a text fileThe LOC:: node (see Psionics file FILEIO) does not distinguish
text and binaryfiles, and always leaves bit 11 clear. Other nodes may distinguish the types of
files and set the bit where appropriate.
Fn $87 Sub $09
FilStatusSet fails vsync
    BX: (cstr) filename
    CX: mask of attributes to be changed (only specify settable bits)    DX: address of the
status word
    DI: new values for attributes to be alteredThe attributes indicated in the mask of the
specified file are altered to
the new values; all other attributes are left unchanged.
Fn $87 Sub $0A AL sync
FilStatusDevice fails vsync
    BX: (cstr) device name (such as "A:" or "LOC::A:")
    CX: 64 byte buffer
    DX: address of the status word
The buffer is filled in with information about the device:  Offset  0 (word): buffer format
version
  Offset  2 (byte): device type
    0 = unknown
    1 = floppy
    2 = hard disc
```

```
    3 = flash
    4 = ram
    5 = rom
    6 = write-protected
  Offset  3 (byte): device properties:
    Bit 3: formattable device
    Bit 4: dual density device
    Bit 5: internal device
    Bit 6: dynamically sizeable device    Bit 7: compressible (worth compressing database
files)
  Offset  4 (word): non-zero if the device contains removable media  Offset  6 (long): total
space on the device in bytes
  Offset 10 (long): free space on the device in bytes
  Offset 14 (cstr): volume name
  Offset 46 (word): device battery status
    0  = low battery
    1  = battery OK
    -4 = status not available for this device type    This field is not valid if the format
version (offset 0) is less than 3.
Fn $87 Sub $0B
FilStatusSystem fails vsync
    BX: (cstr) node name (such as "LOC::" or "REM::")
    CX: 32 byte buffer
    DX: address of the status word
The buffer is filled in with information about the node:  Offset  0 (word): buffer format
version (currently 2)
  Offset  2 (word): 0 = flat, 1 = hierarchical  Offset  4 (word): non-zero if devices are
formattable, and zero otherwise
Fn $87 Sub $0C
FilMakeDirectory fails vsync
    BX: (cstr) pathname
    DX: address of the status word
This is equivalent to the MKDIR keyword.
Fn $87 Sub $0D AL sync
FilOpenUnique fails vsync
   eBX: (cstr) pathname
    CX: open mode (format and access parts only)
```

DX: address of the status wordOpen a file with a unique name. This is equivalent to the IOOPEN keyword with
a mode of 4. The pathname is used to determine the directory to createthe file in, and will be modified to give the actual name (thus the pathname
should have room for 128 characters).
Fn $87 Sub $0E
FilSystemAttach fails vsync
BX: (cstr) name of a file system PDD
DX: address of the status wordThe specified physical device driver will be attached to the filing system,
possible adding new nodes.
Fn $87 Sub $0F
FilSystemDetach fails vsync
BX: (cstr) name of a filing system
DX: address of the status wordDetaches a filing system. Built-in filing systems cannot be detached.
Fn $87 Sub $10
FilPathGetById fails vsync
BX: process to examine
CX: 128 byte buffer
DX: address of the status wordThe buffer is set to the current path of the specified process (a cstr).
Fn $87 Sub $11
FilChangeDirectory fails vsync
BX: (cstr) pathname
CX: 128 byte buffer
DX: address of the status word
SI: (cstr) name of subdirectory DI: 0 = get root directory, 1 = get parent directory, 2 = get subdirectory
The path name is modified in the requested way. This call should be usedinstead of manipulating the directory part of a name directly, as it works onall nodes, irrespective of the format that the node uses for path names.
Fn $87 Sub $12 has no effect on OPL processes.
Fn $87 Sub $13
FilSetFileDate fails vsync
BX: (cstr) pathname
CX: low word of new time

```
    DX: address of the status word
    DI: high word of new time
Sets the modification time of the specified file.
Fn $87 Sub $14
FilLocChanged fails vsync
    AX: -> channel change flags
    BX: channel to check
    DX: address of the status wordSpecifies whether any directory in the LOC:: node has changed
(i.e. a fileor directory has been created, destroyed, or renamed, but not changes to file
contents) since the last use of this system call with this channel. BX shouldhave exactly one
bit set, corresponding to the channel to use; bits 8 to 15are reserved by Psion. The
corresponding bit in AX will be set if the nodehas changed, and will be clear otherwise; the
other 15 bits are unspecified.This call can be used to determine whether to update a directory
display.
Fn $87 Sub $15
FilLocDevice v3 fails vsync
    BX: local device indicator (%A to %H, or %I or %M)    CX: address of word set to the media
type
    DX: address of the status wordProvides the media type of a device on the LOC:: node (I and M
both refer to
the internal ramdisc). The device type consists of:
  Bits 0 to 3:
    0 = unknown
    1 = floppy
    2 = hard disc
    3 = flash
    4 = ram
    5 = rom
    6 = write-protected
  Bits 7 to 8:
    0 = device battery measurement not supported    1 = device battery measurement not
supported
    2 = device battery voltage low
    3 = device battery voltage good
Fn $87 Sub $16
FilLocReadPDD v3 fails vsync
    BX: local device indicator (%A to %H, or %I or %M)
```

```
    CX: (long) location on the device
    DX: address of the status word
    SI: number of bytes to read
    DI: address in process to copy first byte toCopies data from a device on the LOC:: node to
the current process. This
call is very efficient, and accesses the raw device.
Further system calls are described in Psionics file SYSCALLS.2.
```

Revision #1
Created Thu, Jan 24, 2019 10:33 AM by Alex
Updated Thu, Jan 24, 2019 10:33 AM by Alex