SYSCALLS.2

```
PSIONICS FILE - SYSCALLS.2
==========================
System calls (part 2)
Last modified 1998-11-07
=======================
See part 1 for general notes and explanations.
Fn $88 Sub $00
ProcId
    AX: -> process ID of the current process
Gets the process ID of the current process.
Fn $88 Sub $01
ProcIdByName fails
    AX: -> process ID
   eBX: (cstr) patternGets the process ID of a process whose name matches the pattern (usual
wildcards apply, and case is ignored).
Fn $88 Sub $02
ProcGetPriority fails
    AL: -> priority
    BX: process ID
Gets the priority of the specified process.
Fn $88 Sub $03
ProcSetPriority fails
    AL: new priority
    BX: process ID
Sets the priority of the specified process.
Fn $88 Sub $04 should only be called by the operating system.
Fn $88 Sub $05
ProcCreateTask
Tasks are processes which share the data segment of another process. Theycannot be conveniently
handled in OPL.
Fn $88 Sub $06
ProcResume fails
```

```
    BX: process ID
Take a process out of the suspended state and start it executing.
Fn $88 Sub $07
ProcSuspend fails
    BX: process IDPlace a process in the suspended state. If the process is waiting for a system
service (e.g. a semaphore), then it will be suspended when the service has
been carried out.
Fn $88 Sub $08
ProcKill fails
    AL: reason code
    BX: process IDKills the specified process, without allowing it to execute any cleanup code.
Only use this on the current process or in emergencies.
Fn $88 Sub $09
ProcPanicById fails
    AL: panic code
    BX: process ID
Simulate the specified panic on the specified process.
Fn $88 Sub $0A
ProcNameById fails
    BX: process ID
   eDI: 13 byte buffer
Places the name (a cstr) of the specified process in the buffer.
Fn $88 Sub $0B
ProcFind fails
    AX: -> process ID
    BX: 0 or process ID
    SI: 14 byte buffer
   eDI: (cstr) patternObtains process IDs for processes whose name matches the pattern ("?" and
"*"
wildcards have their Unix meaning, and case is ignored). BX should be zero toreturn the first
matching process, or the process ID returned by a previouscall to return subsequent matching
processes. Process are returned in *task*ID order. The buffer is filled with a cstr giving the
process name.
Fn $88 Sub $0C
ProcRename fails
    BX: process ID
   eDI: (cstr) new nameRename the specified process; the new name must be between 1 and 8
```

characters.

Fn $88 Sub $0D

ProcTerminate fails

    BX: process IDTerminates the indicated process. The process will be sent a termination message if it has so requested, and will be killed otherwise.

Fn $88 Sub $0E

ProcOnTerminate

    BX: message typeWhen the current process is terminated, it will be sent a message of the specified type; type 0 cancels the request. After receiving the message andexecuting any clean-up code, the process should use ProcKill to kill itself.

Fn $88 Sub $0F is reserved for the Shell process.

Fn $88 Sub $10

ProcGetOwner fails

    AX: -> owning process ID

    BX: process ID

Gets the ID of the process owning the specified process (normally the creator of that process).

Fn $89 Sub $00

TimSleepForTenths fails

    CX: high half of delay

    DX: low half of delaySleep for the specified delay (in units of 0.1 seconds). Changing the system clock does not affect the call. Only time when the machine is switched on ismeasured; any time when the machine is switched off will be in addition to the requested delay.

Fn $89 Sub $01

TimSleepForTicks fails

    CX: high half of delay

    DX: low half of delaySleep for the specified delay (in system ticks; there are 32 ticks per second on the Series 3 and 18.2 on the PC emulation). Changing the system clock doesnot affect the call. Only time when the machine is switched on is measured;any time when the machine is switched off will be in addition to the requested delay.

Fn $89 Sub $02

TimGetSystemTime

    AX: -> high half of system clock

```
    BX: -> low half of system clock
Reads the system clock (an abstime).
Fn $89 Sub $03
TimSetSystemTime
    CX: high half of new system clock
    DX: low half of new system clock
Sets the system clock to the given abstime.
Fn $89 Sub $04
TimSystemTimeToDaySeconds
    CX: high half of abstime
    DX: low half of abstime
    DI: 8 byte bufferSplits an abstime into a day number and an interval, placed in the buffer as
follows:
  Offset  0 (long): day number
  Offset  4 (long): interval
Fn $89 Sub $05
TimDaySecondsToSystemTime fails
    AX: -> high half of abstime
    BX: -> low half of abstime
    SI: 8 byte bufferConverts a day number and an interval to an abstime. The former are in the
buffer, in the same format as TimSystemTimeToDaySeconds.
Fn $89 Sub $06
TimDaySecondsToDate fails
    SI: 8 byte buffer (day number and interval)
    DI: 8 byte buffer (broken down time)Converts a day number and an interval to broken-down
time information. Theformer is in the same format as TimSystemTimeToDaySeconds. The latter is in
the format:
  Offset  0 (byte): year - 1900
  Offset  1 (byte): month (0 = January, 11 = December)
  Offset  2 (byte): day - 1
  Offset  3 (byte): hours
  Offset  4 (byte): minutes
  Offset  5 (byte): seconds
  Offset  6 (word): day number in year (0 to 364 or to 365)
Fn $89 Sub $07
TimDateToDaySeconds fails
    SI: 8 byte buffer (broken down time)
```

```
    DI: 8 byte buffer (day number and interval)Converts broken-down time to a day number and an
interval. The day number in
year (offset 6) is ignored.
Fn $89 Sub $08
TimDaysInMonth fails
    AX: -> number of days in month
    CX: month * 256 + year - 1900Gets the number of days in the specified month (0 = January, 11
= December).
Fn $89 Sub $09
TimDayOfWeek
    AX: -> day of week (0 = Monday, 6 = Sunday)
    CX: high half of day number
    DX: low half of day number
Gets the day of the week of the given date.
Fn $89 Sub $0A
TimNameOfDay fails
    AL: day of week (0 = Monday, 6 = Sunday)
    BX: 33 byte buffer
The buffer is filled with a cstr giving the name of that day of the week.
Fn $89 Sub $0B
TimNameOfMonth fails
    AL: month (0 = January, 11 = December)
    BX: 33 byte buffer
The buffer is filled with a cstr giving the name of that month.
Fn $89 Sub $0C
TimWaitAbsolute fails
    CX: high half of abstime
    DX: low half of abstimeSleep this process until the specified abstime. If the machine is
turned off
at that time, it will turn back on. Changing the system clock will affect
when the call returns.
Fn $89 Sub $0D
TimWeekNumber fails
    AX: -> week number (1 to 53)
    CX: high half of day number
    DX: low half of day number
Gets the week number of the specified day.
```

Fn $89 Sub $0E

TimNameOfDayAbb v3 fails

AL: day of week (0 = Monday, 6 = Sunday)

BX: 4 byte bufferThe buffer is filled with a cstr giving the abbreviated name of that day of the week. The length of the abbreviated name varies between languages, butis the same for all days in a given language.

Fn $89 Sub $0F

TimNameOfMonthAbb v3 fails

AL: month (0 = January, 11 = December)

BX: 4 byte bufferThe buffer is filled with a cstr giving the abbreviated name of that month. The length of the abbreviated name varies between languages, but is the samefor all months in a given language.

Fn $8A Sub $00

ConvUnsignedIntToBuffer

AX: -> length of converted value

BX: value to be converted

CX: radix

eDI: buffer to hold converted valueThe value is converted to a string in the specified radix and written to the

buffer. No trailing zero byte is written; instead, the length of the stringis returned. The radix can be any value from 2 to 200. If the radix is 11 orgreater, digits greater than 9 are written as "A", "B", etc; characters otherthan digits and uppercase letters are used when the radix is 37 or more.

Fn $8A Sub $01

ConvUnsignedLongIntToBuffer

AX: -> length of converted value

BX: low half of value to be converted

CX: radix

DX: high half of value to be converted

eDI: buffer to hold converted valueThe value is converted to a string, in the same way as ConvUnsignedIntToBuffer.

Fn $8A Sub $02

ConvIntToBuffer

AX: -> length of converted value

BX: value to be converted

eDI: buffer to hold converted valueThe value is converted to a string in radix 10 and written to the buffer. If

the value is negative, a leading "-" will be included. No trailing zero byteis written; instead, the length of the string is returned.

Fn $8A Sub $03

ConvLongIntToBuffer

AX: -> length of converted value

BX: low half of value to be converted

DX: high half of value to be converted

eDI: buffer to hold converted valueThe value is converted to a string in radix 10, as for ConvIntToBuffer.

Fn $8A Sub $04

ConvArgumentsToBuffer

AX: -> length of converted format

BX: buffer holding the format arguments

SI: (cstr) format

eDI: buffer to hold converted formatThe format is written to the buffer, with certain sequences of characters -"conversion specifiers" - being replaced by the values of arguments converted
to strings. No trailing zero byte is written; instead, the length of the string
is returned.

All conversion specifiers begin with a percent sign (to include a literalpercent in the output, use "%%"), followed by:

- an optional alignment code
- an optional width code (required if there is an alignment code)
- an optional length code
- a conversion code.

The alignment code consists of two characters. The first is:- "-": left align (fill on the right)
- "=": centre align (fill at both ends)
- "+": right align (fill on the left).The second is either the character to fill with, or "*". In the latter case,
the next word is taken from the arguments and used as the fill character.The default alignment is to right align filling with spaces ("+ "). Also, the
alignment "+0" may be abbreviated to "0".

The width code gives the number of characters generated by the conversion. Ifthe output would be longer, it is truncated. The code is either an unsigneddecimal number (not beginning with a zero), or "*". In the latter case, thenext word is taken from the arguments and gives the (unsigned) width.

The length code can be "l" or "L" (equivalent); it is equivalent to making theconversion code uppercase.
Each conversion code (apart from "f") takes an argument of the type stated,and then converts it as described.

Code	Type	Conversion	Code	Type	Conversion
"b"	word	convert to unsigned binary representation	"B"	long	convert to unsigned binary representation
"c"	word	output the character with that code	"C"	long	output the character with that code
"d"	word	convert to signed decimal representation	"D"	long	convert to signed decimal representation
"f"	----	output an empty string filled with the fill character.	"F"	----	output an empty string filled with the fill character.
"o"	word	convert to unsigned octal representation			
"O"	long	convert to unsigned octal representation	"m"	word	output the value as 2 bytes, least significant first
"M"	long	output the value as 4 bytes, least significant first			
"s"	cstr	output the cstr			
"S"	cstr	output the cstr			
"u"	word	convert to unsigned decimal representation	"U"	long	convert to unsigned decimal representation
"w"	word	output the value as 2 bytes, most significant first	"W"	long	output the value as 4 bytes, most significant first
"x"	word	convert to unsigned hexadecimal representation	"X"	long	convert to unsigned hexadecimal representationCodes "m", "M", "w", and "W" are available in EPOC v2.17 and later only.

Fn $8A Sub $05
ConvStringToUnsignedInt fails
    AX: -> converted value
    CX: radix
    SI: (cstr) string to convert -> pointer to first unused characterThe string is converted to an unsigned integer in the specified radix. Theconversion ends at the first character which is not a valid digit for theradix, which may be the terminating zero byte; if the first character of the
string is not valid, the call fails. The letters A to F, in either case, areused for digits 10 to 15 if the radix is 11 or more. The radix may be greaterthan 16, but the valid digits remain restricted to the range 0 to 15.

Fn $8A Sub $06

ConvStringToUnsignedLongInt fails

AX: -> high half of converted value

BX: -> low half of converted value

CX: radix

SI: (cstr) string to convert -> pointer to first unused characterThe string is converted, in the same manner as ConvStringToUnsignedInt.

Fn $8A Sub $07

ConvStringToInt fails

AX: -> converted value SI: (cstr) string to convert -> pointer to first unused character The string is converted to an signed integer in radix 10. The conversion endsat the first character which is not a valid digit or leading sign; this maybe the terminating zero byte. If the first character of the string is not a

digit or sign, the call fails.

Fn $8A Sub $08

ConvStringToLongInt

AX: -> high half of converted value

BX: -> low half of converted value SI: (cstr) string to convert -> pointer to first unused character

The string is converted, in the same manner as ConvStringToInt.

Fn $8A Sub $09

ConvFloatToBuffer fails

AX: -> length of converted value

DX: pointer to 6 byte conversion information block SI: address of real value to be converted

DI: buffer to hold converted valueThe real value is converted to a cstr and placed in the buffer. The exactformat of the string depends on the contents of the conversion information block:

Offset 0 (byte): 0 = fixed format, 1 = exponent format, 2 = variable format Offset 1 (byte): maximum length of converted value Offset 2 (byte): number of decimal digits (fixed and exponent formats only)

Offset 3 (byte): radix character

Offset 4 (byte): triad character (fixed format only) Offset 5 (byte): triad threshold (fixed format only)If the converted value would be greater than the maximum length, or the real to be converted has an exponent less than -99 or greater than 99, the callfails and the contents of the buffer are undefined (note that the lengthof the buffer need only be the maximum specified plus an extra byte for the

terminating zero).
For fixed format, the resulting string will take the form: * minus sign if the value is negative; nothing if it is positive * the integer part of the number, with no leading zeros (one zero if the
    integer part is zero); if the triad threshold is non-zero and there are   more than that number of digits in the integer part, then the triad   character is inserted between groups of 3 digits
  * if the number of decimal digits is non-zero:
    - the radix character
    - the specified number of digitsFor example, the number 1234321.4731 is converted as follows (assuming that
the radix character is "." and the triad character is ","):   0 decimal digits, triad threshold 0: "1234321"
    0 decimal digits, triad threshold 5: "1,234,321"   0 decimal digits, triad threshold 7: "1234321"
    2 decimal digits, triad threshold 5: "1,234,321.47"   6 decimal digits, triad threshold 1: "1,234,321.473100"
For floating format, the resulting string will take the form: * minus sign if the value is negative; nothing if it is positive
  * one non-zero digit
  * if the number of decimal digits is non-zero:
    - the radix character
    - the specified number of digits
  * the letter "E"
  * a plus or minus sign
  * two digits (using leading zeros if necessary)For example, the number 1234321.4731 is converted as follows:
    0 decimal digits: "1E+06"
    2 decimal digits: "1.23E+06"
    7 decimal digits: "1.234321E+06"
   12 decimal digits: "1.2343214731 00E+06"Zero (with either sign) is always converted as "0E+00" or "0.000E+00" etc.
For general format the resulting string will take a form depending on itsvalue. If the maximum width is W and the real value is R, then: R <= -1E+(W-1) as for exponent format with W-7 decimal digits -1E+(W-1) < R <= -1E+(W-2) a string containing a minus sign and W-1 digits
  -1E+(W-2) < E <= -1E+(W-1) a string containing a minus sign and W-2 digits -1E+(W-3) < R

<= -1E-4 a string containing a minus sign and then W-2 significant digits with the radix character in the appropriate position -1E-4 < R < 0 as for exponent format with W-7 decimal digits 0 = R the string "0" 0 < R < 1E-4 as for exponent format with W-6 decimal digits 1E-4 <= R < 1E+(W-2) a string containing W-1 significant digits with the radix character in the appropriate position 1E+(W-2) <= E < 1E+(W-1) a string containing W-1 digits 1E+(W-1) <= E < 1E+(W) a string containing W digits 1E+(W) <= R as for exponent format with W-7 decimal digits

Fn $8A Sub $0A

ConvStringToFloat fails

DX: radix character

SI: address of a word pointing to the start of the cstr to convert DI: address of real variable to be setConverts a string representing a floating-point value and places it in the variable. The string must have the form:

* an optional sign * a mantissa containing at least one digit and an optional radix character
* an optional exponent consisting of:
  - the letter "E" or "e"
  - an optional sign
  - an integer

The resulting value must have an exponent in the range -99 to 99 inclusive.If it is outside this range, the call fails; for underflow the value 0 isstored, while for overflow an undefined value is stored.

If the call succeeds, the pointer to the string is altered to point to thefirst character beyond the converted number.

Fn $8B Sub $00

GenVersion

AX: -> kernel versionGets the version of the operating system. Version 1.23 is reported as:

$123A for alpha release

$123B for beta release

$123F for final release

Fn $8B Sub $01

GenLcdType

```
    AL: -> display type
Gets the display type:
     0 = 640x400 LCD (MC)
     1 = 640x200 LCD small version (MC)     2 = 640x200 LCD large version or CGA (PC
emulation)
     3 = 720x348 LCD or Hercules graphics
     4 = 160x80  LCD (HC)
     5 = 240x80  LCD (Series 3t)
     6 = MDA (PC emulation)
     7 = EGA monochrome (PC emulation)
     8 = EGA colour     (PC emulation)
     9 = VGA monochrome (PC emulation)
    10 = VGA colour     (PC emulation)
    11 = 480x160 LCD (Series 3a and Series 3c)
    12 = 240x100 LCD (Workabout)
    14 = 240x160 LCD (Siena)
   255 = unknown
Fn $8B Sub $02
GenStartReason
    AL: -> reason code
Gets the reason for the last cold start:
    0 = system RAM invalid
    1 = forced power down
    2 = user reset (using reset button)
    3 = kernel fault
    4 = new operating system installedThe environment variables and the internal disc are valid
after reasons 1 and
3, and are valid after reason 2 unless ESC was also pressed.
Reason 1 only happens if a faulty device driver delayed a normal battery-lowpowerdown for too
long.
Fn $8B Sub $03
GenParse fails
    BX: 18 byte bufferParse filenames according to certain basic rules. Unlike FilParse, this
does
not invoke any device drivers. The buffer has the format:  Offset  0 (word): address of file
name 1 (a cstr)
  Offset  2 (word): address of file name 2 (a cstr)  Offset  4 (word): address of file name 3 (a
```

cstr)

Offset 6 (word): address of buffer to be filled with final name (a cstr) Offset 8 (word): address of 6 byte buffer

Offset 10 (byte): device separator

Offset 11 (byte): path separator

Offset 12 (byte): extension separator Offset 13 (byte): maximum length of device including separators

Offset 14 (byte): maximum length of path including separators Offset 15 (byte): maximum length of name

Offset 16 (byte): maximum length of extension including separatorThe three filenames are split into 5 components, any of which may be missing.

* node (ends with "::")
* device (ends with device separator)
* path (ends with last path separator)
* name
* extension (starts with extension separator)The final name is constructed by taking, for each component, the value fromfile name 1 if present, otherwise the value from file name 2 if present, and otherwise the value from file name 3. The 6 byte buffer is then filled inwith the same data as for FilParse. File names 2 and 3 should be in different
places in memory.

Fn $8B Sub $04

LongUnsignedIntRandom

AX: -> high half of random number BX: address of seed -> low half of random number

Generates a 32 bit unsigned random number from a seed; the number also replacesthe seed, so that the sequential calls with the same seed address will generatea random sequence based on the initial seed.

Fn $8B Sub $05

GenGetCountryData

BX: 40 byte buffer

Fills the buffer with country-specific data: Offset 0 (word): country code (e.g. UK is 44) of locale

Offset 2 (word): current offset from GMT in minutes (+ is ahead) Offset 4 (byte): date format (0 = MDY, 1 = DMY, 2 = YMD)

Offset 5 (byte): time format (0 = am-pm, 1 = 24 hour) Offset 6 (byte): currency symbol position (0 = before, 1 = after) Offset 7 (byte): currency symbol separated with space (0 = yes, 1 = no)

Offset 8 (byte): currency decimal places Offset 9 (byte): currency negative format (0 =

minus, 1 = brackets)
  Offset 10 (byte): currency triad threshold
  Offset 11 (byte): triad separator
  Offset 12 (byte): decimal separator
  Offset 13 (byte): date separator
  Offset 14 (byte): time separator
  Offset 15 to 23: currency symbol (cstr) Offset 24 (byte): start of week (0 = Mon, 1 = Tue, ... 6 = Sun)
  Offset 25 (byte): currently active summer times:
    Bit 0:   home
    Bit 1:   European
    Bit 2:   Northern
    Bit 3:   Southern
    Bits 4 to 7: unused, always zero
  Offset 26 (byte): clock type (0 = analogue, 1 = digital) Offset 27 (byte): number of letters in day abbreviation (0 to 6) Offset 28 (byte): number of letters in month abbreviation (0 to 255)
  Offset 29 (byte): workdays (the set bits indicate the workdays)
    Bit 0: Monday
    Bit 1: Tuesday
    Bit 2: Wednesday
    Bit 3: Thursday
    Bit 4: Friday
    Bit 5: Saturday
    Bit 6: Sunday
    Bit 7: always zero
  Offset 30 (byte): units (0 = inches, 1 = centimetres)
If the triad threshold is non-zero and there are more than that number ofdigits in the integer part of an amount of money, then the triad charactershould be inserted between groups of 3 digits. See ConvFloatToBuffer for a
way to do this.
Fn $8B Sub $06
GenGetErrorText
    AL: error number
    BX: 64 byte bufferThe buffer will be filled with a cstr giving an error message corresponding to
the error number (a generic message including the number is used when there isno stored message

in the current locale).

Fn $8B Sub $07

GenGetOsData

CX: number of bytes to fetch

SI: offset of first byte in kernel workspace

DI: buffer to be filled inCopy a number of bytes from the kernel workspace to the current process.

Within assembler code, this can also be done via GenDataSegment (see Psionics file KERNEL).

Fn $8B Sub $08 only applies to MC systems.

Fn $8B Sub $09

GenNotify fails

AL: -> option chosen (0 = first, 1 = second, 2 = third) BX: first message (cstr, up to 63 characters plus the terminating zero) CX: second message (cstr, up to 63 characters plus the terminating zero) DX: first option (cstr, up to 15 characters plus the terminating zero)

SI: second option (cstr, up to 15 characters plus the terminating zero) DI: third option (cstr, up to 15 characters plus the terminating zero)Sends a message to the notifier process and waits for a reply. The call failsif there is no notifier process running. The two messages are displayed, and

the user is offered the three options. The chosen option is returned.

Any of the arguments except BX may be zero instead; for CX this means thatonly one message is displayed, while for DX, SI, and DI it means that lessoptions are offered (if all are zero, the option "CONTINUE" is offered).

Fn $8B Sub $0A

GenNotifyError fails

AL: error number -> option chosen (0 = first, 1 = second, 2 = third)

BX: first message

DX: first option

SI: second option

DI: third optionThis call is equivalent to GenNotify with the second message derived from the error number via GenGetErrorText.

Fn $8B Sub $0B and $0C are used by the notifier process.

Fn $8B Sub $0D

GenGetRamSizeInParas

AX: -> system RAM sizeGets the amount of system RAM fitted, in units of 16 bytes. Note that this

is an unsigned value - $8000 represents 512 kb, not some negative amount.

Fn $8B Sub $0E

GenGetCommandLine

    AX: -> zero or command lineGets a pointer to the command line if the program was started with program

information (see FilExecute). If so, the command line consists of a cstr givingthe program executed, immediately followed by the program information (a qstr).

Fn $8B Sub $0F

GenGetSoundFlags

    AX: -> sound flags

Gets the sound flags:

    Bit 0: keyboard clicks enabled

    Bit 1: sound via piezo buzzer enabled

    Bit 2: sound via SND: device driver enabled   Bit 3: keyboard clicks: set=loud, clear=soft

    Bit 4: piezo buzzer: set=loud, clear=soft   Bit 15: set=disable all sound, clear=obey individual flags

Fn $8B Sub $10

GenSetSoundFlags

    BX: sound flags

Sets the sound flags to new values (see GenGetSoundFlags).

Fn $8B Sub $11

GenSound

    BX: duration in ticks

    CX: pitch code (frequency=512kHz/code)Makes a simple single-frequency note. Access to this call is sequenced; itwill wait until the piezo buzzer is not in use, and will return when the sound

has started to play. On the Series 3a, the piezo buzzer is emulated by theSND: device, but the emulation is complete (for example, this call workseven when SND: is disabled by GenSetSoundFlags).

Fn $8B Sub $12

GenMarkActive

Fn $8B Sub $13

GenMarkNonActive

These two calls alter the state of the current process to "active" (the defaultwhen the process starts) or "non-active". Whenever an active process restartsexecution after being idle (basically when keywords such as IOWAIT, IOW, GETand so on return), the auto-off timer is reset;

the machine switches off when
the timer reaches the appropriate value.
Fn $8B Sub $14
GenGetText fails
    AL: message number
   eBX: 64 byte buffer
Copies a message (a cstr) from the kernel message table to the buffer.
Fn $8B Sub $15
GenGetNotifyState
    AL: -> notify state (0=unattended, 1=notify)Gets the notify state for the process. If this is "unattended", then the fileserver will not call the notifier process when a correctable error occurs (such
as an SSD with open file being removed), but will simply return an error. Ifit is "notify" (the initial state), then it will use GenNotify to request the
user to fix the problem.
Fn $8B Sub $16
GenSetNotifyState
    AL: notify state
Sets the notify state for the process. See GenGetNotifyState for details.
Fn $8B Sub $17
GenGetAutoSwitchOffValue
    AX: -> auto-off time in seconds, or $FFFF if disabled
Gets the auto-off time.
Fn $8B Sub $18
GenSetAutoSwitchOffValue
    BX: auto-off time in seconds, or $FFFF to disableSets the auto-off time. Times of less than 15 are adjusted to 15.
Fn $8B Sub $19 and $1A are for device drivers only.
Fn $8B Sub $1B
GenGetLanguageCode
    AX: -> current locale codeGets the current locale code. Locale codes are listed in the Psionics file
LOCALES.
Fn $8B Sub $1C
GenGetSuffixes
   eBX: 93 byte bufferThe buffer is filled with 31 cstrs giving the correct suffix for each of the

31 days of the month. The suffix for day N is at offset (3*N-3).

Fn $8B Sub $1D

GenGetAmPmText

AL: 0=AM, 1=PM

eBX: 3 byte bufferThe buffer is filled with a cstr giving the correct suffix for "a.m." or "p.m." times.

Fn $8B Sub $1E

GenSetCountryData

eBX: 40 byte bufferSets the country-specific data to that in the buffer. See GenGetCountryData for the format of the buffer.

Fn $8B Sub $1F

GenGetBatteryType

AL: -> battery type

Gets the battery type from the following list:

0 = Unknown

1 = Alkaline

2 = NiCaD (600 mAh)

3 = NiCaD (1000 mAh)

4 = NiCaD (500 mAh)

The battery type is used to control the warning thresholds.

Fn $8B Sub $20

GenSetBatteryType

AL: battery type

Sets the battery type (see GenGetBatteryType).

Fn $8B Sub $21

GenEnvBufferGet fails

AX: -> size of value

DX: length of pattern

eSI: buffer filled with value

eDI: pattern to search forSearches for an environment variable whose name matches the pattern (if there

is more than one, which one is chosen is unspecified). The buffer is filledwith the variable's value.

Fn $8B Sub $22

GenEnvBufferSet fails

CX: size of value (0 to 255)

DX: length of name (1 to 16)

```
   eSI: buffer holding value
   eDI: name of variableChanges the value of the specified enviromment variable, creating it
first if
necessary. The name may not contain wildcards.
Fn $8B Sub $23
GenEnvBufferDelete fails
    DX: length of name
   eDI: pattern to deleteSearches for an environment variable whose name matches the pattern and
deletes
it (if there is more than one, which one is deleted is unspecified).
Fn $8B Sub $24
GenEnvBufferFind fails
    AX: -> new handle
    BX: previous handle (0 for first call)
    DX: length of pattern
   eSI: 273 byte buffer
   eDI: pattern to search forSearches for each environment variable whose name matches the
pattern. The
call is made multiple times until it fails; the same pattern must be usedeach time. If the call
succeeds, the buffer is filled with two qstrs, thefirst being the name and the second the value
of the variable found.
Fn $8B Sub $25
GenEnvStringGet fails
   eSI: 256 byte buffer
   eDI: pattern (cstr)Searches for an environment variable whose name matches the pattern (if
there
is more than one, which one is chosen is unspecified). The buffer is filledwith the variable's
value followed by a zero byte (thus a cstr).
Fn $8B Sub $26
GenEnvStringSet fails
   eSI: value (cstr)
   eDI: name (cstr)Changes the value of the specified enviromment variable, creating it first if
necessary. The name may not contain wildcards and is limited to 16 characters;the value is
limited to 255 characters.
Fn $8B Sub $27
GenEnvStringDelete fails
   eDI: pattern (cstr)Searches for an environment variable whose name matches the pattern and
```

deletes
it (if there is more than one, which one is deleted is unspecified).
Fn $8B Sub $28
GenEnvStringFind fails
    AX: -> new handle
    BX: previous handle (0 for first call)
   eCX: 256 byte buffer
   eSI: 17 byte buffer
   eDI: pattern to search forSearches for each environment variable whose name matches the pattern. The
call is made multiple times until it fails; the same pattern must be usedeach time. If the call succeeds, the 17 byte buffer is filled with the nameof the variable, as a cstr, and the 256 byte buffer is filled with the value
followed by a zero byte (thus also a cstr).
Fn $8B Sub $29
GenCrc
    AX: -> CRC
    CX: buffer length
    DX: previous CRC
    SI: bufferCalculates the CCITT Cyclic Redundancy Checksum using the X^16+X^12+X^5+1 polynomial (e.g. the CRC of a single byte with value 1 is $1021). If thebuffer is to be checked standalone, use zero as the previous checksum. Ifseveral blocks are to be checksummed as if they were one long block, use 0as the previous CRC for the first block, and the previous result for the
remainder.
Fn $8B Sub $2A
GenRomVersion
    AX: -> rom version numberGets the version of the system ROM (which includes both the operating system
and many files). Version 1.23 is reported as:
    $123A for alpha release
    $123B for beta release
    $123F for final release
Fn $8B Sub $2B and $2C are used by the alarm server process.
Fn $8B Sub $2D
GenAlarmId
    AX: -> alarm server pidGets the process ID of the alarm server process, or 0 if none is

running.
Fn $8B Sub $2E
GenPasswordSet fails
   uSI: (qstr) current password
   uDI: (qstr) new passwordSets a new system password. The call will fail and take no action if the
current password is incorrect.
Fn $8B Sub $2F
GenPasswordTest fails
   uSI: (qstr) password to test
Succeeds if the system password is that provided.
Fn $8B Sub $30
GenPasswordControl fails
    AL: new state (0 = off, 1 = on)
   uSI: (qstr) current passwordTurns the system password on and off. The call will fail and take no action
if the current password is not provided.
Fn $8B Sub $31
GenPasswordQuery
    AX: -> new state (0 = off, 1 = on)
Get the status of the system password.
Fn $8B Sub $32
GenTickle
Resets the auto-off timer. A non-active process (see GenMarkNonActive) canuse this to prevent auto-off.
Fn $8B Sub $33
GenSetConfig
@No documentation available at present@
Fn $8B Sub $34
GenMaskInit fails
    SI: (qstr) password
    DI: 18 byte encryption control blockInitializes an encryption control block according to the password given.
The contents of the control block for the current algorithm are: Offset 0 to   8: encryption key, generated from the password
  Offset 9 to 15: copy of offset 0 to 7 Offset 16 (word): current location within the key (initially zero)

@The algorithm for generating the key from the password is unknown.
It is possible that future versions of this call may implement multipleencryption algorithms, so AL, BX, CX, and DX - unused at present - should
be set to zero.
Fn $8B Sub $35
GenMaskEncrypt fails
    CX: length of buffer
    SI: encryption control block
    DI: buffer holding data to be encryptedEncrypts a block of data according to the encryption control block, whichwill be updated. The value of the encrypted data depend only on the original data and on the contents of the control block, which should normally beinitialized by GenMaskInit. Provided that the control block is not modifiedin any other way, encrypting a block of data in two or more parts yields the
same result as encrypting it all at once.
The current encryption algorithm operates on a byte by byte basis as follows.For each data byte in turn, take the value of offset 16 in the control block(which will be from 0 to 15) and use it to select one of the first 16 bytesin the control block (that is, offset 16 = 0 selects offset 0, offset 16 = 1
selects offset 1, and so on). Add the selected byte to the data byte modulo$100. Finally, add 1, modulo 16, to offset 16 of the control block.
An example of this algorithm in use can be found in the file WORD.FMT.
It is possible that future versions of this call may implement multipleencryption algorithms, so AL, BX, and DX - unused at present - should be set
to zero.
Fn $8B Sub $36
GenMaskDecrypt fails
    CX: length of buffer
    SI: encryption control block
    DI: buffer holding data to be decryptedDecrypts a block of data according to the encryption control block, which
will be updated. The data should have been encrypted with GenMaskEncryptwith the same initial control block. All comments applying to the latter call
apply here.
Fn $8B Sub $37
GenSetOnEvents v2.28
    AL: 0=disable 1=enableEnables or disables reporting of power-on events. By default this is enabled

on the Series 3 and disabled on other systems. Disabling may affect other applications.

Fn $8B Sub $38

GenGetAutoMains v3

AX: -> settingGets the setting of the external power auto-off flag. Zero means that auto-off will take place even if external power is present, while non-zero means thatauto-off is suspended while external power is present.

Fn $8B Sub $39

GenSetAutoMains v3

AL: new settingSets the external power auto-off flag. Zero means that auto-off will take place even if external power is present (unless it has been disabled bysetting the auto-off time to $FFFF), while non-zero means that auto-off is suspended while external power is present.

Fn $8C Sub $00

FloatSin fails [sine]

Fn $8C Sub $01

FloatCos fails [cosine]

Fn $8C Sub $02

FloatTan fails [tangent]

Fn $8C Sub $03

FloatASin fails [arc sine]

Fn $8C Sub $04

FloatACos fails [arc cosine]

Fn $8C Sub $05

FloatATan fails [arc tangent]

Fn $8C Sub $06

FloatExp fails [exponentiation (base e)]

Fn $8C Sub $07

FloatLn fails [natural logarithm (base e)]

Fn $8C Sub $08

FloatLog fails [decimal logarithm (base 10)]

Fn $8C Sub $09

FloatSqrt fails [square root]

SI: argument (real)

DI: result (real)Calculates the specified function of the argument. The argument and result may be stored in the same place; if not, the argument is not altered. Thetrignometric functions operate in radians.

```
Fn $8C Sub $0A
FloatPow fails
    DX: power (real)
    SI: base (real)
    DI: result (real)Calculates the result of raising the base to the indicated power. The result
may be stored in the same place as either argument; if not, neither argument
is altered.
Fn $8C Sub $0B
FloatRand
    SI: seed (long)
    DI: result (real)Generates a random real number based on the unsigned long integer seed,
which
is unaltered; the memory used for the two should not overlap.
Fn $8C Sub $0C
FloatMod fails
    DX: divisor (real)
    SI: dividend (real)
    DI: result (real)
Calculates the dividend modulo the divisor, using the formula:    result = dividend - divisor *
FloatInt (dividend/divisor)The result may be stored in the same place as either argument; if
not, neither
argument is altered.
Fn $8C Sub $0D
FloatInt fails
    SI: argument (real)
    DI: result (real)Rounds the argument to the closest integer towards zero (i.e. zeros the
fractional part of the argument). The argument and result may be stored in thesame place; if
not, the argument is not altered.
Further system calls are described in Psionics file SYSCALLS.3.
```

Revision #1
Created Thu, Jan 24, 2019 10:34 AM by Alex
Updated Thu, Jan 24, 2019 10:34 AM by Alex