WILDLIFE ACOUSTICS

SMART System

API and Command Reference

2025-01-01

Terms of Use

Customer agrees to be bound by the terms and conditions of Wildlife Acoustics, Inc., which can be found at https://www.wildlifeacoustics.com/legal-documentation. Customer further agrees that any End User is made aware of and bound by such terms and conditions.

Introduction

The following descriptions of the SMART daemon, command-line scripts, and SMART library APIs are from the manual pages bundled with the SMART System software. This information can be used for additional customized programming of the SMART system.

You can read this same content from the SMART System command line using the `man` command. For example, `man smart-bat-sim` displays the same content as smart-bat-sim *(on page 5)*.

Contents

Chapter 1. Executable Programs

smart-bat-sim

A Bat Alarm simulator for generating Pass and Pulse events

Synopsis

`smart-bat-sim [OPTIONS]...`

Description

smart-bat-sim is a test tool for generating Bat Pass & Pulse alarms. The simulator will generate either Pass or Pulse events and post them into the Kaleidoscope shared memory event queue. Once queued, the system will process the event using the defined filters from the SCADA configuration. Events are generated at a rate as dictated by the alarm-period, high-water-mark and low-water-mark. The high water rate is calculated as **(alarm-period / high water + 1)** - so with a high water of 20 events within a period of 60 seconds the sim generates an event every 2 seconds. Note that we add an extra event to ensure we meet the time period and account for any variances. The same formula is used to calculate the low water rate.

The sim will start generating events at the calculated rate to reach the high water mark. At that point, you should see the event '**raised**' in the smart-scada-log.txt file. For example:

2024-01-19,10:12:57,1,pulse,alarm1 pulse,raised

Once the number of events generated reaches the high water mark, the sim starts to generate events at the low water rate. Then, there is a final delay simply waiting enough time for the '**clear**' condition to be reached - in the logfile you should see: 2024-01-19,10:13:57,1,pulse,alarm1 pulse,cleared

This cycle is repeated for the desired number of iterations, where one iteration is a raise/clear combination.

Options

--alarm-period*=*alarm-period

The pass/pulse time period. Default 60 seconds

--high-water*=*high-water

The Pass/Pulse high water mark. Default 20 events

--iterations*=*iterations

The Number of iterations (raise/clear cycles) to run. Default 2 iterations

--low-water*=*low-water

The Pass/Pulse low water mark. Default 5 events

--type*=*type

The Type of event to generate {pass | pulse}. Default pulse

--verbose

Set this option for more verbose messages

-?, --help

Give this help list

--usage

Give a short usage message

--version

Print program version

Exit Codes

0

Success

1

Fail

Related information
smart-scada *(on page 11)*

smart-check-filter

check a SMART SCADA filter specification

Synopsis

`smart-check-filter` ***`string`***

Description

Check the syntax of the filter expression provided as a single command line argument. See the SMART documentation for more information.

Exit Codes

0

Expression is valid

1

Expression is not valid

Related information
smart-scada *(on page 11)*

smart-ctl

control SMART devices

Synopsis

`smart-ctl [` ***`sleep`*** `|` ***`wake`*** `|` ***`reset`*** `|` ***`upgrade`*** `]` ***`serial-number`***

Description

Request a given SMART device identified by the 6-octet hexadecimal serial-number change state to either sleep, wake, reset. or upgrade

Files

If upgrade is specified, the file /usr/local/share/smart/firmware_%d where %d is the device model number is sent to the device for firmware upgrade. This file is typically a symbolic link to an actual version-specific file.

Related information
smart-stream *(on page 17)*
smart-daemon *(on page 7)*
smart-list *(on page 8)*

smart-daemon

daemon process to interface with SMART devices

Synopsis

`smart-daemon [--force] [-i ifname ]`

Description

Binds to the specified Ethernet interface (or the first suitable interface if ifname is not specified) and begins communication with SMART devices over Ethernet. A control socket is created to interface between the daemon and user processes by means of the libSMART shared library.

The SMART daemon must be run as root. A valid license is required to run the SMART daemon and to use the libSMART libraries.

Options

--force

is used to force the SMART daemon to run even if a PID file already exists.

-i interface

specifies the Ethernet device interface to use. Otherwise, the first suitable interface found is used automatically.

Files

/var/run/smart/smart_pid

A file containing the process id of the currently running SMART daemon.

/var/run/smart/smart_ctl

A UNIX socket used by libSMART to communicate with the SMART dameon.

/usr/local/share/smart/smart.lic

A license file required for using the SMART daemon and libSMART.

Related information
smart-stream *(on page 17)*
SMART_Close *(on page 23)*
SMART_CloseWav *(on page 23)*
SMART_CreateWav *(on page 24)*
SMART_GetDeviceInfo, SMART_GetDeviceInfoBySN, SMART_GetNDevices *(on page 25)*
SMART_GetSerial *(on page 28)*
SMART_KaleidoscopeAnalyzeBlock256 *(on page 29)*
SMART_KaleidoscopeAnalyzeFlush *(on page 30)*
SMART_KaleidoscopeAnalyzeGetZCFile *(on page 31)*
SMART_KaleidoscopeCreate *(on page 32)*
SMART_KaleidoscopeDelete *(on page 35)*
SMART_Open *(on page 38)*
SMART_Read *(on page 40)*
SMART_Reset *(on page 41)*
SMART_Sleep *(on page 43)*

SMART_Wake *(on page 44)*
SMART_Write *(on page 45)*
SMART_WriteWav *(on page 45)*

smart-dispatcher

SMART Microphone Scheduler

Synopsis

```
smart-dispatcher
```

Description

smart-dispatcher runs as a service and schedules attached microphone devices for streaming. For each scheduled microphone, **smart-dispatcher** spawns an instance of **smart-stream-wrapper**.

The **smart-dispatcher** scans the devices directory /var/www/html/storage/devices/ for device-specific subdirectories named with their unique serial number (six dotted hex bytes). Within each subdirectory, the dispatcher looks for a settings and schedule configuration file. If both files are present, the microphone device is scheduled in accordance with the schedule configuration file. When scheduled to record, the **smart-stream-wrapper** is invoked with the --duration parameter set in accordance to the scheduled end time, and other parameters are passed in accordance to the settings configuration file.

The settings configuration file is a list of arguments to be passed to **smart-stream-wrapper**, each argument on a line.

The schedule configuration file is a text file comprising a list of schedule blocks. Each schedule block is three lines as follows:

START { **TIME** | **RISE** {+|-} | **SET** {+|-} } hh:mm

DUTY { **ALWAYS** | **CYCLE ON** hh:mm **OFF** hh:mm }

END { **TIME** | **RISE** {+|-} | **SET** {+|-} } hh:mm

Exit Codes

0

Success

1

Fail

Files

/var/www/html/storage/devices/ *<serial-number>* **/schedule**

Per-microphone schedule configuration

Related information

smart-daemon *(on page 7)*
smart-stream-wrapper *(on page 21)*
smart-stream *(on page 17)*

smart-list

list SMART devices

Synopsis
`smart-list`

Description

Lists detected SMART devices and their capabilities.

Related information
smart-stream *(on page 17)*
smart-daemon *(on page 7)*

smart-logger

Periodically log system information for SMART services

Synopsis
`smart-logger [ -i <seconds> ]`

Description

The SMART Logger runs as a service and periodically monitors disk, CPU, and memory utilization as well as battery voltage levels for other SMART services to use (e.g. for serial outputs, SCADA/Modbus, etc). The default interval is 10 seconds unless otherwise specified on the command line.

For battery voltage, a bash shell script /usr/local/bin/SaveVoltage.sh is executed. This script should write the battery voltage, in volts, to the file /tmp/LastVoltage. The value from this file is then read by the smart-logger service or could be used by other applications.

Files

/usr/local/bin/SaveVoltage.sh as described above

/tmp/LastVoltage as described above

Related information
smart-serial *(on page 15)*
smart-scada *(on page 11)*

smart-modbus-probe

Read or Write local Modbus registers via smart-scada.

Synopsis
`smart-modbus-probe modbus-address [ count [ value, ... ] ]`

Description

Read or write local Modbus registers. See the SMART documentation for the Modbus register map which includes registers managed by the smart-scada service as well as registers available for custom applications.

Parameters

modbus-address

6-digit decimal Modbus register starting address

count

If specified, the number of sequential registers to read and write.

values...

If specified, the number of values must match count. Values are 16-bit signed or unsigned values to be written to sequential Modbus registers beginning with the modbus-address. If not specified, the specified registers are read and printed as signed 16-bit values, one per line.

Exit Codes

0

Success

1

Fail

Related information
smart-scada *(on page 11)*

smart-opcua

SMART OPC UA Service

Synopsis

`smart-opcua`

Description

smart-opcua runs as a service and provides a secure mechanism for retrieving status information, system updates and alarm notifications from a SMART device using OPC UA communications. OPC UA clients are able to take appropriate actions according to local needs, including bat curtailment processing. The Information Model used by the SMART Controller provides a flexible and comprehensive set of parameters, methods and publish/subscribe features to enable full monitoring of a SMART system. The parameters exposed by smart-opcua are mirrored from the Modbus registers available from the smart-scada service.

smart-opcua will only start if the file /var/www/storage/scada.json exists. This file is created when SCADA has been configured from the SMART Control Panel.

Options

Startup options can be modified in the **/etc/defaults/smart-opcua.opts** file. Simply edit the file and add the option(s) needed to the SMART_OPCUA_OPTS setting.

--port=port number

The TCP port for smnart-opcua to listen. Default 4840

--certificate=certificate file

Full path to the X.509 certificate .der file. Default /etc/ssl/private/server_cert.der

--key=private key file

Full path to the private key .pem file. Default /etc/ssl/private/server_key.pem

--secure-channel-trust-folder=channel trust folder

Full path to the secure channel trust certs. Default /etc/ssl/private/trust

--session-trust-folder=sesion trust folder

Full path to the secure channel trust certs. Default /etc/ssl/private/trust

Exit Codes

0

Success

1

Fail

Files

/var/www/html/storage/scada.json

The SCADA Configuration file

/etc/defaults/smart-opcua.opts

Configuration file used to override smart-opcua startup options

/usr/local/share/smart/opcua/smartModel.xml

The SMART Information Model defiinition

/usr/local/share/smart/opcua/smartModel.csv

The Information Model node ID definitions

/usr/local/share/smart/opcua/Smart.NodeSet2.xml

The compiled smartModel.xml into a NodeSet2 format ingested by the smart-opcua server.

Related information
smart-scada *(on page 11)*

smart-scada

SMART SCADA/Modbus Service

Synopsis

```
smart-scada
```

Description

smart-scada runs as a service and responds to Modbus requests over either Modbus/TCP (over TCP/IP networks) or Modbus/RTU (over serial interfaces). Real-time bat pulse/pass information is collected from the SMART event infrastructure as well as system status information from the SMART logger to update Modbus registers accordingly.

The SMART SCADA system can be configured to use either Modbus/TCP or Modbus/RTU. For Modbus/RTU, additional serial parameters must be specified. Up to eight (8) filters can be defined (numbered one (1) through eight (8)). Each filter can be applied to individual bat echolocation pulses or entire bat passes as they are reported to the SMART event system.

The scada.json configuration file is a JSON formatted file with the following structure:

modbusPort

String indicating either "TCP" for Modbus/TCP or a TTY e.g. "ttyS0" for Modbus/RTU. An empty string indicates that the SMART SCADA service is disabled.

modbusSlaveId

For RTU protocol, this is the server ID (numeric)

modbusBaudRate

For RTU protocol, this is the baud rate (numeric)

modbusParity

For RTU protocol, this string indicates parity as one of "None", "Even", or "Odd".

modbusDataBits

For RTU protocol, this is the number of data bits 5, 6, 7, or 8.

modbusStopBits

For RTU protocol, this is the number of stop bits 1 or 2.

filters

An array of active filters:

filters[N].filterNumber

The filter nubmer one (1) through eight (8) defined by this array element.

filters[N].eventPeriod

Integer number of rolling seconds for bat pass rate monitoring.

filters[N].eventHighWater

Integer bat pass rate threshold triggering the filter bat pass alarm.

filters[N].eventLowWater

Integer bat pass rate threshold clearing the filter bat pass alarm.

filters[N].pulsePeriod

Integer number of rolling seconds for bat pulse rate monitoring.

filters[N].pulseHighWater

Integer bat pulse rate threshold triggering the filter pulse pass alarm.

filters[N].pulseLowWater

Integer bat pulse rate threshold clearing the filter pulse pass alarm.

filters[N].spec

Filter specificiation string (see below)

Filter Specification

The filter specificiation is a boolean expression that can match bat call parameters (and microphone device names as defined by their "prefix"). As individual echolocation calls match a filter expression, the bat pulse period, high-water mark and low-water mark are used to raise or clear a per-filter bat pulse alarm. Similarly, as sequences of echolcoation calls forming a bat pass (as defined by smart-stream triggering parameters match a filter expression, the bat pass period, high-water mark and low-water mark are used to raise or clear a per-filter bat pass alarm.

An expression can be a numeric comparison between floating point values. Values can be specified as floating point literals e.g. -1.23 or any of the bat call parameters **N**, **Fc**, **Sc**, **Fmax**, **Fmin**, **Fmean**, **TBC**, **Fk**, **Tk**, **S1**, **Tc Dur**, or **Qual**. Comparison operators can be = for equality, **!=** or **<>** for not equal, **<** for less than, **<=** for less than or equal, **>** for greater than, or **>=** for greater than or equal.

A floating point value can also be calculated using addition **+** or subtraction **-** between floating point literals and bat call parameters.

In order to differentiate among multiple microphone devices, the variable **prefix** can be compared to a string literal using = for equality, **!=** or **<>** for not equal, or **~** if the left hand side contains the right hand side as a substring. String literals are text surrounded by matching single or double quotes.

For multi-channel microphones, the variable **channel** can be compared to a numeric channel number. Note that at this time, the only SMART microphone SMART-MIC-1 has only one channel (0).

Boolean expressions can be combined using the operators **AND**, **OR** or inverted with **NOT**.

Nested parentheses can be used as per normal convention.

A comment can be indicated with any text following the **#** character.

Note that white space is ignored, filter expressions can span multiple lines, and variable names are case insensitive.

Modbus Register Map

The following table describes the Modbus register map. Registers marked as reserved for customization can be read or written using smart-modbus-probe.

Function	Address	Description
2	10000**N**	Filter #**N** bat pass alarm
2	10001**N**	Filter #**N** bat pulse alarm
2	10002**X**	Reserved for customization
3/6/16	4000**N**1	Filter #**N** bat pass alarm period
3/6/16	4000**N**2	Filter #**N** bat pass high-water mark
3/6/16	4000**N**3	Filter #**N** bat pass low-water mark
3/6/16	4000**N**4	Filter #**N** bat pulse alarm period
3/6/16	4000**N**5	Filter #**N** bat pulse high-water mark
3/6/16	4000**N**6	Filter #**N** bat pulse low-water mark
3/6/16	40009**X**	Reserved for customization
4	300001	Disk utilization (%)
4	300002	Memory utilization (%)
4	300003	CPU utilization (%)
4	300004	Battery voltage (0.1V)
4	3000**N**1	Filter #**N** bat pass event counter
4	3000**N**2	Filter #**N** bat pass events per sec-ond
4	3000**N**3	Filter #**N** bat pass time since last event (s)
4	3000**N**4	Filter #**N** bat pass alarm counter
4	3000**N**5	Filter #**N** bat pulse event counter
4	3000**N**6	Filter #**N** bat pulse events per sec-ond
4	3000**N**7	Filter #**N** bat pulse time since last event (s)

Function	Address	Description
4	3000**N**8	Filter #**N** bat pulse alarm counter
4	300**N**01	Filter #**N** bat pass most recent N
4	300**N**02	Filter #**N** bat pass most recent Fc
4	300**N**03	Filter #**N** bat pass most recent Sc
4	300**N**04	Filter #**N** bat pass most recent Fmax
4	300**N**05	Filter #**N** bat pass most recent Fmin
4	300**N**06	Filter #**N** bat pass most recent Fmean
4	300**N**07	Filter #**N** bat pass most recent TBC
4	300**N**08	Filter #**N** bat pass most recent Fk
4	300**N**09	Filter #**N** bat pass most recent Tk
4	300**N**10	Filter #**N** bat pass most recent S1
4	300**N**11	Filter #**N** bat pass most recent Tc
4	300**N**12	Filter #**N** bat pass most recent Dur
4	300**N**13	Filter #**N** bat pass most recent Qual
4	3009**XX**	Reserved for customization

Event Hook

If a file smart-scada-hook.sh exists, it will be executed as a bash script when an alarm condition changes. The first argument is the alarm number (1-8). The second argument is the alarm type "pass" or "pulse", and the third argument is 1 to indicate the alarm was activated or 0 to indicate the alarm was cleared.

Exit Codes

0

Success

1

Fail

Files

/var/www/html/storage/scada.json

Configuration file

/var/www/html/storage/smart-scada-hook.sh

Hook file

Related information

smart-modbus-probe *(on page 9)*
smart-logger *(on page 9)*
smart-serial *(on page 15)*

smart-serial

SMART Serial Service

Synopsis

`smart-serial`

Description

smart-serial runs as a service and writes bat pass events and periodic status data to the specified serial port as events are collected from the SMART event infrastructure.

The serial.conf configuration file contains 8 lines with the following structure:

1. **TTY**

   The serial port to use e.g. "ttyS0".

2. **{7|8}{N|E|O}{1|2}<baudrate>**

   Specifies the data bits, parity, and baudrate e.g. "8N19600" indicates 8 data bits, no parity, 1 stop bit, and 9600 baud.

3. **mask**

   The mask is a 32-bit hexadecimal number indicating which output fields to include lines written to the serial port. See below for details.

4. **event line preamble**

   Characters to send at each event line start. \r and \n can be used to indicate carriage-return and new-line characters respectively.

5. **event line epilogue**

   Characters to send at each event line end including line termination. \r and \n can be used to indicate carriage-return and new-line characters respectively.

6. **polling interval**

   Polling interval for output of periodic status information in seconds, or zero to disable periodic status information output. **7. status line preamble** Characters to send at each status line start. \r and \n can be used to indicate carriage-return and new-line characters respectively.

8. **status line epilogue**

   Characters to send at each status line end including line termination. \r and \n can be used to indicate carriage-return and new-line characters respectively.

The following table describes the mask bits (hex values or'ed together):

Mask Bits	Description
0x00000001	Prefix
0x00000002	Date
0x00000004	Time
0x00000008	Duration
0x00000010	N
0x00000020	Fc
0x00000040	Sc

Mask Bits	Description
0x00000080	Fmax
0x00000100	Fmin
0x00000200	Fmean
0x00000400	TBC
0x00000800	Fk
0x00001000	Tk
0x00002000	S1
0x00004000	Tc
0x00008000	Dur
0x00010000	nPulsesClassified
0x00020000	nPulsesMatching
0x00040000	Top1Match
0x00080000	Top1Margin
0x00100000	Top2Match
0x00200000	Top2Margin
0x00400000	Top3Margin
0x00800000	Top3Margin
0x01000000	Event generated CRC7
0x02000000	Status Date
0x04000000	Status Time
0x08000000	Status Disk Utilization (%)
0x10000000	Status Events since last poll
0x20000000	Voltage
0x40000000	Reserved
0x80000000	Status generated CRC7

Exit Codes

0

Success

1

Fail

Files

/var/www/html/storage/serial.conf

Configuration file

Related information
smart-modbus-probe *(on page 9)*
smart-logger *(on page 9)*
smart-serial *(on page 15)*

smart-stream

stream and process audio from SMART device microphone device

Synopsis

`smart-stream [REQUIRED PARAMETER]... [OPTION]...`

Description

Opens an audio stream from a SMART device and optionally creates audio files and/or analyzes the data.

Required Parameters

--microphone serial number

Device serial number, hexadecimal with 6 octets (12 hex digits)

--sample -rate sample rate

Sample rate in Hz matching device capabilites

Run options

--duration duration

Specify the duration to run the stream in seconds, or in mm:ss or hh:mm:ss format. After this duration, the program exits. If not specified, the program will run indefinitely until the connection to the device is lost.

Device options

--channel channel number

Specify the channel number 0 - 7. The default channel is 0 if not specified.

--high-pass high pass filter

High-pass filter in Hz matching device capabilities.

--gain gain

Gain in dB matching device capabilities.

--use-backup-sensor

Specify using backup sensor if available in device capabilities. Otherwise the primary sensor will be used.

--enable-heater

Specify using defogging heater if available in device capabilities. Otherwise the heater will be turned off.

--no-heater-analysis

If used with ***--enable-heater*** above, the data stream will not be analyzed. Specifically, this option overrides and disables ***--bats*** and ***--non-bats*** and there will be no bat triggers or events.

--calibrate frequency amplitude

Generate a calibration signal at the specified frequency (Hz) and amplitude (1.0 full scale). As the microphone signal is returned, the dB level of the calibration signal (narrow-band filtered) is displayed on exit.

--stdout path

Append standard output (csv results) to specified file.

--stderr path

Append standard error (logging) to specified file.

File creation options

Audio files representing each event (if analysis is enabled) or the audio stream for the specified duration can be created. The filename is of the form

prefix_YYYYMMDD_hhmmss_uuuuuu.ext

where .ext is either .wav, .w4v or .zc.

--output-wav directory

If specified, timestamped .wav or .w4v files will be created in the directory specified.

--output-zc directory

If specified, timestamped .zc files will be created in the directory specified. This option requires either ***--bats*** or ***--non-bats***.

--compression compression

If ***--output-wav*** is specified, compression values 4, 6 or 8 can be used to specify Wildlife Acoustics .w4v format with the specified numer of bits per sample. The default is 0 which indicates no compression and creation of standard .wav files.

--prefix prefix

Specify a prefix string that is pre-pended to the filename.

--pre-trigger pre-trigger

Specify the amount of extra time in seconds to include in the output recording prior to the first detected signal.

--post-trigger post-trigger

Specify the amount of extra time in seconds to include in the output recording after the last detected signal (this does not apply to .zc files.

--location latitude longitude

Specify the latitude and longitude to include in metadata.

--guano string

Specify additional lines of GUANO metadata key:value pairs.

Kaleidoscope Analysis Options

--bats **or** ***--non-bats***

Specify bat analysis mode or non-bat analysis mode. Only one mode can be enabled. One of these options is required to enable analysis.

***--min-freq** min frequency*

Specify the minimum frequency of the expected signal in Hz. The default is 8000 Hz if ***--bats*** is specified and 250 Hz if ***--non-bats*** is specified.

***--max-freq** max frequency*

Specify the maximum frequency of the expected signal in Hz. The default is 120000 Hz if ***--bats*** is specified and 10000 Hz if ***--non-bats*** is specified.

***--min-dur** min duration*

Specify the minimum pulse duration (if ***--bats*** is specified) or detection duration (if ***--non-bats*** is specified) in seconds. The default is 0.002 if ***--bats*** is specified or 0.100 if ***--non-bats*** is specified.

***--max-dur** max duration*

Specify the maximum pulse duration (if ***--bats*** is specified) or detection duration (if ***--non-bats*** is specified) in seconds. The default is 0.500 if ***--bats*** is specified or 7.500 if ***--non-bats*** is specified.

***--max-gap** max gap*

Specify the maximum inter-pulse gap (if ***--bats*** is specified) or inter-detection gap (if ***--non-bats*** is specified) in seconds. The default is 0.500 if ***--bats*** is specified or 0.350 if ***--non-bats*** is specified.

***--max-sequence** max sequence*

Specify the maximum detection event duration (if ***--bats*** is specified) in seconds. The default is 15.0 seconds.

***--min-pulses** min pulses*

Specify the minimum number of pulses required to consider a signal a bat (if ***--bats*** is specified). The default is 2.

***--cf-filt-max-freq** constant frequency filter max frequency*

Specify the maximum frequency in Hz for the constant frequency filter. The default is off (0).

***--cf-filt-max-bw** constant frequency filter max bandwidth*

Specify the maximum bandwidth in Hz for the constant frequency filter. The default is off (0).

--disable-enhanced-zc

Disable the enhanced zero-crossing algorithm used by Kaleidoscope. Instead of more sophisticated narrow-band analysis, the signal will be band-pass filtered by the min frequency and max frequency and then undergo zero-cross analysis.

***--classifier** classifier*

A classifier can be specified. This can be either a cluster .kcs file or a directory containing the unzipped files from a Bat Auto ID classifier.

***--threshold** threshold*

A threshold can be specified to adjust the sensitivity of the classifier. For bat auto id classifiers, this value is either -1 for "more sensitive", 0 for "balanced" or 1 for "more accurate". For cluster analysis, this is the maximum distance, a floating point value between 0 and 2. The default is 1.

***--species** species list*

A comma separated list of species codes to include in the classifier. If not specified, all classifier species codes are included.

Outputs

If ***--calibrate*** is specified, the dB level (re 0 dB = full scale) is output in a one-line message narrow-band filtered on the given frequency.

If ***--output-wav*** is specified, timestamped .wav or .w4v files will be generated. If ***--output-zc*** is specified, timestamped .zc files will be specified.

If analysis mode is enabled (e.g. with either ***--bats*** or ***--non-bats***), a line of text will be output for each detection. Each line has a list of comma separated text fields as follows:

YYYY-MM-DD

Date representing the beginning of the event (not adjusted for pre-trigger)

hh:mm:ss.uuuuuu

Time representing the beginning of the event (not adjusted for pre-trigger)

Duration

Duration of detected event in seconds

N

For ***--bats***, number of pulses detected. The following statistics are measurements averaged over these pulses.

Fc

For ***--bats***, characteristic frequency, Hz

Sc

For ***--bats***, characteristic slope, octaves per second

Fmax

Maximum frequency, Hz

Fmin

Minimum frequency, Hz

Fmean

Mean frequency, Hz

TBC

For ***--bats***, time between calls, seconds

Fk

For ***--bats***, frequency of the knee, Hz

Tk

For ***--bats***, time of the knee, seconds

S1

For ***--bats***, initial slope, octaves per second

Tc

For ***--bats***, time of characteristic, seconds

Dur

For ***--bats***, pulse duration, seconds

nPulsesClassified

For ***--bats***, number of pulses classified

nPulsesMatching

For ***--bats***, number of pulses matching final classification

Top1Match

First matching classification label

Top1Margin

First matching classification margin

Top2Match

Second matching classification label

Top2Margin

Second matching classification margin

Top3Match

Third matching classification label

Top3Margin

Third matching classification margin

Related information
smart-daemon *(on page 7)*
smart-stream-wrapper *(on page 21)*

smart-stream-wrapper

A wrapper for smart-stream for scheduling

Synopsis

```
smart-stream-wrapper [OPTIONS]...
```

Description

smart-stream-wrapper adds a maintenance schedule layer around smart-stream to manage automatic multi-sensor calibration, selection, and schedule a duty-cycle for activating a built-in heater element. smart-stream-wrapper will invoke the smart-stream process sequentially to conform with the maintenance schedule based on the additional maintenance options described below until terminated by SIGINT or error. Other options are passed to the smart-stream verbatim with the exception of ***--duration***. If ***--duration*** is specified, this is the maximum value passed to the underlying smart-stream process. The smart-stream-wrapper may use shorter values to conform to the maintenance schedule.

The smart-stream-wrapper will selectively pass ***--enable-heater***, and ***--use-backup-sensor*** in accordance with the maintenance schedule. For calibration, smart-stream-wrapper will invoke smart-stream using ***--calibrate*** to measure the response of the two redundant sensors and automatically choose the best one.

Options

--schedule-heater on-duration off-duration

Specify the on-duration and off-duration duty-cycle for the heater in seconds. The --enable-heater parameter will be passed to the underlying smart-stream program during the on periods.

--schedule-calibrate freq amp target period

A calibration will run every period seconds to calibrate and measure the primary sensor (for one second) and the backup sensor (for another second) by invoking smart-stream with the --calibrate option. The frequency in Hz and the amplitude as a fraction of full scale is passed to smart-stream. The sensor measuring closest to the target value in dB relative to full scale will be chosen. Subsequent invocations of smart-stream will optionally use --use-backup-sensor.

--stdout path

Append standard output (csv results) to specified file.

--stderr path

Append standard error (logging) to specified file.

Hook

When smart-stream-wrapper performs a calibration sequence, a user-defined bash shell script may be executed to respond to microphone communciation failures and calibration measurements. If the file /var/www/html/storage/smart-calibrate-hook.sh is present, it will be executed with the following arguments:

serial-number ($1)

Device serial number.

primary result ($2)

Primary sensor calibration result, or blank on error.

backup result ($3)

Backup sensor calibration result, or blank on error.

result target ($4)

Target result as passed from smart-stream-wrapper, or blank on error.

Files

/var/www/html/storage/smart-calibrate-hook.sh Optional hook invoked during calibration cycle

Related information
smart-stream *(on page 17)*

Chapter 2. Library Functions

SMART_Close

close audio stream with a SMART device such as the calibratoion transducer of a microphone.

Synopsis

```
#include <smart.h>

int SMART_Close(int streamid);
```

Description

Given a streamid returned from a previous call to **SMART_Open(),** close the stream.

Return Value

SMART_Close() returns the number of bytes written from buffer or -1 on error setting errno.

Errors

EINVAL

Invalid streamid

EBUSY

Bad state

Related information
smart-daemon *(on page 7)*
SMART_Open *(on page 38)*
SMART_Read *(on page 40)*
SMART_Write *(on page 45)*
SMART_GetDeviceInfo, SMART_GetDeviceInfoBySN, SMART_GetNDevices *(on page 25)*

SMART_CloseWav

close a .wav or .w4v file previously opened with **SMART_CreateWav().**

Synopsis

```
#include <smart.h>

int SMART_CloseWav(int handle);
```

Description

Given handle returned from a previous call to **SMART_CreateWav()** complete writing out and closing the file.

Return Value

SMART_CloseWav() Returns 0 on success or -1 on error setting errno.

Errors

EINVAL

Invalid handle

EIO

An error occurred trying to write data to the file.

EBUSY

Bad state

Related information
SMART_CreateWav *(on page 24)*
SMART_WriteWav *(on page 45)*

SMART_CreateWav

create a .wav or .w4v file.

Synopsis

```
#include <smart.h>
#include <time.h>

int SMART_CreateWav(struct smart_wav_info_s * infop);
```

Description

Given parameters indicated by infop open a WAV or W4V file for writing.

Parameters for creating the .wav or .w4v file are indicated by a smart_wav_info_s structure as specified in <smart.h>:

```
struct smart_wav_info_s
  {
    const char *path;       // filename
    int         nchannels;  // number of channels
    int         samplerate; // sample rate, Hz
    int         compression;// compression mode
    struct timespec timestamp; // timestamp of metadata
    double      latitude;   // latitude for metadata
    double      longitude;  // longitude for metadata
    const char *guano;      // additional GUANO or NULL
  };
```

The **SMART_CreateWav()** function creates and opens a file indicated by path for writing by subsequent calls to **SMART_WriteWav().** The nchannels indicates the number of channels of interleaved audio are present. The compression indicates either an uncompressed PCM .WAV file if 0, or a compressed Wildlife Acoustics .W4V file specifying 4, 6, or 8 bits per sample. Other parameters are used to create the necessary file headers and include additional GUANO formatted meta information. If guano is non-null, it points to an optional multi-line string of additional GUANO key:value pairs for user-defined meta-data in addition to standard metadata.

Return Value

SMART_CreateWav() returns a stream handle, a small non-negative integer number, used in subsequent calls to **SMART_WriteWav()** and **SMART_CloseWav().** On error, -1 is returned and errno is set.

Errors

EINVAL

Invalid compression value. Must be 0 for uncompressed .wav, or 4, 6, or 8 for .w4v.

EIO

An error occurred trying to write file headers.

EACCESS

Permission denied trying to create or open the file specified by path.

Related information

SMART_WriteWav *(on page 45)*
SMART_CloseWav *(on page 23)*

SMART_GetDeviceInfo, SMART_GetDeviceInfoBySN, SMART_GetNDevices

get information about SMART devices such as microphones.

Synopsis

```
#include <smart.h>

int SMART_GetNDevices();

int SMART_GetDeviceInfo(int device, struct smart_device_info_t *infop);

int SMART_GetDeviceInfoBySN(uint8_t *sn, struct smart_device_info_t *infop);
```

Description

The **SMART_GetNDevices()** function caches information about detected SMART devices such as microphones from the SMART daemon and returns a count N indicating the number of devices discovered.

Subsequent calls to **SMART_getDeviceInfo()** return information about one of the discovered devices from 0 - N -1.

The **SMART_GetDeviceInfoBySN()** returns information about a specific device identified by a 6 octet serial number (mac address) without cacheing.

Device information is returned in a smart_device_info_t structure as sepcified in <smart.h>:

```
struct smart_device_info_t
  {
    uint8_t sn[6];     // serial number (MAC address) of device
    uint8_t model;     // model number
    uint8_t nchannels; // number of channels
    uint32_t capabilities; // bitmask of capabilities (see below)
    uint8_t is_sleeping;  // true if device is asleep
    uint8_t is_streaming; // true if device is streaming
    uint64_t rx_samples; // receive samples
    uint64_t rx_seq_err; // receive sequence errors
    uint64_t rx_drops;   // receive samples dropped due to congestion
    uint64_t tx_samples; // transmit samples (calibration data)
    uint32_t tx_pkts;    // dev eth transmit packet counter
    uint32_t tx_cols;    // dev eth transmit with collission counter
```

```
    uint32_t rx_pkts;    // dev eth receive packet counter
    uint32_t rx_crc;     // dev eth receive CRC error counter
    uint32_t rx_align;   // dev eth recieve alignment error counter
};
```

The capabilities is a bit mask of device capabilities defined in <smart_device_modes.h> defining supported sample rates, high-pass filter settings, gains settings, sensor selection, calibration, and heating.

SMART_SR_MASK

Sample rates:

SMART_SR_8KHZ

8,000 Hz

SMART_SR_12KHZ

12,000 Hz

SMART_SR_16KHZ

16,000 Hz

SMART_SR_22KHZ

22,050 Hz

SMART_SR_24KHZ

24,000 Hz

SMART_SR_32KHZ

32,000 Hz

SMART_SR_44KHZ

44,100 Hz

SMART_SR_48KHZ

48,000 Hz

SMART_SR_96KHZ

96,000 Hz

SMART_SR_192KHZ

192,000 Hz

SMART_SR_384KHZ

384,000 Hz

SMART_SR_500KHZ

500,000 Hz

SMART_HP_MASK

High-pass filter settings:

SMART_HP_OFF

No high-pass filter set

SMART_HP_250HZ

250 Hz

SMART_HP_1KHZ

1 kHz

SMART_HP_8KHZ

8 kHz

SMART_HP_16KHZ

16 kHz

SMART_GAIN_MASK

Gain settings

SMART_GAIN_0DB

0 dB

SMART_GAIN_6DB

6 dB

SMART_GAIN_12DB

12 dB

SMART_GAIN_18DB

18 dB

SMART_GAIN_24DB

24 dB

SMART_GAIN_30DB

30 dB

SMART_GAIN_36DB

36 dB

SMART_GAIN_42DB

42 dB

SMART_CH_0

Has a primary microphone sensor

SMART_CH_1

Has a secondary (backup) microphone sensor

SMART_CH_CAL

Has a calibration transducer

SMART_HEATER

Has a heating element (for defogging the sensors)

Return Value

SMART_GetNDevices() returns the number of devices detected or -1 on error setting errno.

SMART_GetDeviceInfo() and **SMART_TetDeviceInfoBySN()** return 0 on success or -1 on error setting errno

Errors

ECONNREFUSED

Unable to connect to the SMART daemon

EACCESS

Permission denied from missing or incorrect license, or unable to connect to SMART daemon socket.

ENODEV

Information about the requested device could not be found

EINTR

Error occurred while communicating with SMART daemon.

Bugs

The **SMART_GetNDevices()** and **SMART_GetDeviceInfo()** functions are not thread safe as they cache a query from the SMART daemon in a static variable.

Related information
smart-daemon *(on page 7)*
SMART_Reset *(on page 41)*
SMART_Wake *(on page 44)*
SMART_Sleep *(on page 43)*

SMART_GetSerial

Get serial number of SMART system

Synopsis

```
#include <smart.h>

int SMART_GetSerial(uint8_t *sn);
```

Description

The **SMART_GetSerial()** function write the 6 octet serial number (mac address) of the SMART system to sn by way of the SMART daemon. This is the MAC address of the Ethernet interface bound by the SMART daemon for communicating with devices such as microphones.

Return Value

SMART_GetSerial() return 0 on success or -1 on error setting errno.

Errors

ECONNREFUSED

Unable to connect to the SMART daemon

EACCESS

Permission denied from missing or incorrect license, or unable to connect to SMART daemon socket.

EINTR

Error occurred while communicating with SMART daemon.

Related information
smart-daemon *(on page 7)*
SMART_GetDeviceInfo, SMART_GetDeviceInfoBySN, SMART_GetNDevices *(on page 25)*

SMART_GetSystemStatus

Read system status from logger

Synopsis

```
#include <smart.h>

intSMART_GetSystemStatus(structsmart_system_status*status);
```

Description

Reads the system status from the logger service.

```
// System status
struct smart_system_status
  {
    time_t timestamp;// last update
    size_t bavail;   // available blocks
    size_t blocks;   // total blocks
    double voltage;  // voltage reading
    double load;     // load average
    size_t memtotal; // total memory (k)
    size_t memavail; // available memory (k)
  };
```

Return Value

Returns 0 on success, -1 otherwise.

Errors

No errors have yet been defined (always returns success).

Related information
SMART_KaleidoscopeCreate *(on page 32)*
SMART_KaleidoscopeAnalyzeFlush *(on page 30)*
SMART_KaleidoscopeAnalyzeGetZCFile *(on page 31)*
SMART_KaleidoscopeDelete *(on page 35)*
SMART_KaleidoscopeEventInit *(on page 36)*
SMART_KaleidoscopeEventPost *(on page 37)*
SMART_KaleidoscopeEventNext *(on page 36)*
SMART_GetSystemStatus *(on page 29)*
SMART_SetSystemStatus *(on page 42)*

SMART_KaleidoscopeAnalyzeBlock256

pass 256 samples to a Kaleidoscope analysis object for processing

Synopsis

```
#include <smart.h>

int SMART_KaleidoscopeAnalyzeBlock256( Kaleidoscope Handle handle, const in16_t * samples);
```

Description

Given a Kaleidosocpe Analysis instance identified by handle returned from a previous call to **SMART_KaleidoscopeCreate(),** pass 256 16-bit audio samples (512 bytes) from samples. If a detected event occurs, the callback function provided by **SMART_KaleidoscopeCreate()** may be called zero or more times, once for each detected event.

Return Value

Returns 0 on success, -1 otherwise.

Errors

No errors have yet been defined (always returns success).

Related information
SMART_KaleidoscopeCreate *(on page 32)*
SMART_KaleidoscopeAnalyzeFlush *(on page 30)*
SMART_KaleidoscopeAnalyzeGetZCFile *(on page 31)*
SMART_KaleidoscopeDelete *(on page 35)*
SMART_KaleidoscopeEventInit *(on page 36)*
SMART_KaleidoscopeEventPost *(on page 37)*
SMART_KaleidoscopeEventNext *(on page 36)*

SMART_KaleidoscopeAnalyzeFlush

Complete processing of buffered data

Synopsis

```
#include <smart.h>

int SMART_KaleidoscopeAnalyzeFlush( Kaleidoscope Handle handle);
```

Description

Given a Kaleidosocpe Analysis instance identified by handle returned from a previous call to **SMART_KaleidoscopeCreate(),** complete any processing of buffered data. samples. If a detected event occurs, the callback function provided by **SMART_KaleidoscopeCreate()** may be called zero or more times, once for each detected event.

Return Value

Returns 0 on success, -1 otherwise.

Errors

No errors have yet been defined (always returns success).

Related information
SMART_KaleidoscopeCreate *(on page 32)*
SMART_KaleidoscopeAnalyzeBlock256 *(on page 29)*

SMART_KaleidoscopeAnalyzeGetZCFile

Extract ZC file from detection event.

Synopsis

```
#include <smart.h>

intSMART_KaleidoscopeAnalyzeGetZCFile
 ( Kaleidoscope Handle  handle
        ,const char *  filename
        ,uint8_t *  buffer
        ,size_t  length
        ,struct timespec *  timestamp
        ,double  latitude
        ,double  longitude
        ,const char *  guano
        ,double  pretrigger
        );
```

Description

The function **SMART_KaleidoscopeGetZCFile()** may be called from the callback function specified in a previous call to **SMART_KaleidoscopeCreate().** The callback is typically invoked when the application calls either SMART_KaleidoscopeAnalyzeBlock256() or SMART_KaleidoscopeAnalyzeFlush(). This function writes the contents of a .zc file to the buffer specified up to length bytes in length. For alignment, the file provides pretrigger seconds ahead of the first detected signal. Other parameters including timestamp, latitude, longitude, and guano are used to provide additional meta data to the file.

A detection can include a zero-crossing representation in "Bat Analysis Mode", or a trace of the peak detected signal throughout a vocalization in "Non-Bat Analysis Mode" which can be rendered in a .zc zero-crossing file. The .zc file format is an extremely compact representation of an acoustic detection event representing a single frequency point for each point in time where signal is detected.

Return Value

Returns the number of bytes written to buffer or -1 on error setting errno.

Errors

ENOBUFS

The buffer provided was not large enough to receive the zero crossing data.

EIO

Unable to determine the SMART system serial number.

Notes

With metadata, a typical .zc file can fit within 65,536 bytes. The size of the file depends on the signal content of the detection.

Related information
SMART_KaleidoscopeCreate *(on page 32)*
SMART_KaleidoscopeAnalyzeBlock256 *(on page 29)*
SMART_KaleidoscopeAnalyzeFlush *(on page 30)*
SMART_KaleidoscopeDelete *(on page 35)*
SMART_KaleidoscopeEventInit *(on page 36)*
SMART_KaleidoscopeEventPost *(on page 37)*
SMART_KaleidoscopeEventNext *(on page 36)*

SMART_KaleidoscopeCreate

Create a Kaleidsocope instance for analysis of a single-channel audio stream.

Synopsis

```
#include <smart.h>

KaleidoscopeHandle SMART_KaleidoscopeCreate(struct smart_kaleidoscope_params * params,
 smart_kaleidoscope_event_callback callback);
```

Description

Given a set of parameters indicated by the structure params, create an instance of a Kaleidoscope analysis channel for analyzing an audio stream. The callback function specified will be called back from subsequent calls of **SMART_KaleidoscopeAnalyzeBlock256()** and **SMART_KaleidoscopenalyzeFlush()** in response to detected acoustic events.

The analysis parameters are provided in a smart_kaleidosocpe_params structure as follows:

```
struct smart_kaleidoscope_params
  {
    // Kaleidoscope analysis mode
    enum
      {
        MODE_BATS     = 0
       ,MODE_NON_BATS = 1
      } mode;          // Kaleidoscope analysis mode

    int samplerate;    // input samplerate, Hz

    // Analysis signal parameters
    //   Note: If using a cluster classifier, these values are
    //   inherited from the .kcs file.
    double minFreq;    // minimum frequency, Hz
    double maxFreq;    // maximum frequency, Hz
    double minDur;     // minimum duration, s
    double maxDur;     // maximum duration, s
    double maxGap;     // maximum inter-syllable gap, s

    // For bat analysis, the maximum sequence duration can be
    // specified to force the end of a trigger if maxGap isn't
    // observed.
    double maxSequence;// maximum duration of bat sequence, s

    // For bat analysis, minimum number of pulses required
    int    minPulses;   // bat mode minimum pulses
    int    zcEnhance;   // bat mode use enhanced processing
```

```
    // Classifier:
    //   Null for using signal detector
    //   .kcs file for cluster classifier
    //   .wcl file for old autoid bat classifiers
    //        (3.1.0 and earlier)
    //   directory for new autoid bat classifiers
    //        (4.1.0 and later) unzipped
    const char *path;

    // for autoid <0=sensitive,0=balanced,>0=accurate
    // for cluster classifier, this is max dist
    double      thold;

    // list of species codes with comma delimeters
    // or NULL for all
    const char *species;

    // constant frequency noise filter parameters
    double cfFilterMaxFreq;       // max frequency Hz to apply filter
    double cfFilterMaxBandwidth;  // max bandwidth Hz to apply filter

    // Prefix (microphone name)
    const char *prefix;
  };
```

Kaleidosocpe analysis has two distinct modes including "Bat Analysis Mode" and "Non-Bat Analysis Mode". In "Bat Analysis Mode", the signal parameters minFreq, maxFreq, minDur, and maxDur describe the range of expected echolocation pulses within a sequence while maxSequence describes the maximum duration of a sequence of pulses within a "detection". The minPulses specifies the minimum number of echolocation pulses in a sequence to be considered as a bat vs. noise. And zcEnhance should be true to indicate use of Kaleidoscope's advanced signal processing to convert full spectrum signals to zero crossing signals. Otherwise a broadband filter (between minFreq and maxFreq is applied and the signal is zero-crossed without further processing.

In "Non-Bat Analysis Mode" minFreq, maxFreq, minDur, and maxDur describe the range of expected "detections".

In both modes, the maxGap parameter indicates the maximum inter-pulse gap (for "Bat Analysis Mode") or detection gap (for "Non-Bat Analysis Mode") to determine when a detection event ends and a new detection event begins.

A bat auto-id classifier can be specified when using "Bat Analysis Mode" by setting path to point to a directory containing the un-zipped files comprising an Auto ID classifier from Kaleidoscope. The thold parameter indicates "more sensitive" (-1), "balanced" (0), or "more accurate" (1). A cluster classifier can be specified (in either "Bat Analysis Mode" or "Non-bat Analysis Mode" by specifying a .kcs file with path and the thold parameter indicates the maximum distance for a successful classification on a scale of 0 - 2. If a bat auto-id classifier or cluster classifier is specified, all species are selected by default unless species is non-null, in which case a comma separated list of species codes can be provided and only those species listed will be considered by the classifiers.

Data is passed to the classifier via **SMART_KaleidoscopeAnalyzeBlock256()** and ultimately flushed at the end of the data stream with **SMART_KaleidoscopeFlush().** These functions can in turn invoke the provided callback function for each detected event. The callback function has the following prototype:

```
typedef void (*smart_kaleidoscope_event_callback)
              (KaleidoscopeHandle handle
              ,struct smart_kaleidoscope_results *results
              );
```

The results describing the detected event are provided in the following smart_kaleidoscope_results structure:

```
enum KaleidoscopeResultType
  {
    RESULT_TYPE_PASS
   ,RESULT_TYPE_CALL
  };

struct smart_kaleidoscope_results
  {
    // Detection:
    //   The Kaleidoscope analysis engine buffers future samples
    //   while analyzing data in the past.  Therefore, when an
    //   event is detected, it may be after several subsequent
    //   blocks of samples have been passed to Kaleidoscope.
    //
    //   pre_offset indicates the negative offset (in seconds)
    //   prior to the last data fed to
    //   SMART_KaleidoscopeAnalyzeBlock256() where this detected
    //   event begins.
    double pre_offset; // negative offset to beginning, s
    double duration;   // duration of detected event, s

    // Autoid or clustering
    const char *ids[3];     // top 3 classificiation results
    float       margins[3]; // top 3 margin results

    // If autoid for bats
    int         nPulsesClassified; // number of pulses classified
    int         nPulsesMatching;   // number pulses matching id

    // Data specific to bat analysis
    // (not valid for non-bat analysis except Fmax, Fmin, Fmean)
    // These are generally averages across each detected
    // echolocation pulsein the detected trigger event.
    int    N;     // number of detected echolocation pulses
    double Fc;    // average characteristic frequency, Hz
    double Sc;    // average characteristic slope, octaves per second
    double Fmax;  // average maximum frequency, Hz
    double Fmin;  // average minimum frequency, Hz
    double Fmean; // average mean frequency, Hz
    double TBC;   // average time between calls, seconds
    double Fk;    // average frequency at the knee, Hz
    double Tk;    // average time at knee, seconds
    double S1;    // average initial slope, octaves per second
    double Tc;    // average time of characteristic, seconds
    double Dur;   // average pulse duration, seconds
    double Qual;  // average pulse quality, %

    // Prefix populated by libsmart from the smart_kaleidoscope_params
#   define SMART_RESULTS_MAX_PREFIX 32
    char prefix[SMART_RESULTS_MAX_PREFIX];

    // These for event logging, not populated by Kaleidoscope libraries
    uint32_t seq;         // used internally by logger to detect wrap
    struct timespec timestamp; // timestamp, optional, set by client
    uint8_t sn[6];        // serial number, optional, set by client
```

```
    // Result type indicated by SMART library
    enum KaleidoscopeResultType resultType;
  };
```

Return Value

SMART_KaleidoscopeCreate() returns a handle of type KaleidoscopeHandle or NULL on error setting errno.

Errors

EACCESS

Permission denied from missing or incorrect license or unable to open classifier file if specified.

EIO

Error occurred trying to read classifier file if specified.

Related information
SMART_KaleidoscopeDelete *(on page 35)*
SMART_KaleidoscopeAnalyzeBlock256 *(on page 29)*
SMART_KaleidoscopeAnalyzeFlush *(on page 30)*
SMART_KaleidoscopeAnalyzeGetZCFile *(on page 31)*
SMART_KaleidoscopeEventInit *(on page 36)*
SMART_KaleidoscopeEventPost *(on page 37)*
SMART_KaleidoscopeEventNext *(on page 36)*

SMART_KaleidoscopeDelete

Delete an instance of Kaleidoscope analysis.

Synopsis

```
#include <smart.h>

int SMART_KaleidoscopeDelete( Kaleidoscope Handle handle);
```

Description

Given a Kaleidosocpe Analysis instance identified by handle returned from a previous call to **SMART_KaleidoscopeCreate(),** delete the instance and free resources.

Return Value

Returns 0 on success, -1 otherwise.

Errors

No errors have yet been defined (always returns success).

Related information
SMART_KaleidoscopeCreate *(on page 32)*
SMART_KaleidoscopeAnalyzeBlock256 *(on page 29)*
SMART_KaleidoscopeAnalyzeFlush *(on page 30)*
SMART_KaleidoscopeAnalyzeGetZCFile *(on page 31)*
SMART_KaleidoscopeEventInit *(on page 36)*
SMART_KaleidoscopeEventPost *(on page 37)*
SMART_KaleidoscopeEventNext *(on page 36)*

SMART_KaleidoscopeEventInit

Initialize event subsystem for posting or retrieving events

Synopsis

```
#include <smart.h>

intSMART_KaleidoscopeEventInit();
```

Description

Connect to the SMART Kaleidoscope Event system for posting or retrieving events. This function must be called before calling **SMART_KaleidoscopeEventPost(), SMART_KaleidoscopeEventNext(), SMART_GetSystemStatus(),** or **SMART_SetSystemStatus().**

Return Value

Returns 0 on success, -1 otherwise.

Errors

No errors have yet been defined (always returns success).

Related information
SMART_KaleidoscopeCreate *(on page 32)*
SMART_KaleidoscopeAnalyzeFlush *(on page 30)*
SMART_KaleidoscopeAnalyzeGetZCFile *(on page 31)*
SMART_KaleidoscopeDelete *(on page 35)*
SMART_KaleidoscopeEventInit *(on page 36)*
SMART_KaleidoscopeEventPost *(on page 37)*
SMART_KaleidoscopeEventNext *(on page 36)*
SMART_GetSystemStatus *(on page 29)*
SMART_SetSystemStatus *(on page 42)*

SMART_KaleidoscopeEventNext

Get next event for specified consumer

Synopsis

```
#include <smart.h>

int SMART_KaleidoscopeEventNext( enum KaleidoscopeEventConsumer consumer
 , struct smart_kaleidoscope_results *event
 , int block
 , struct timespec *timeout);
```

Description

Get the next event for the specified consumer. The following consumers are defined:

```
 KALEIDOSCOPE_EVENT_CONSUMER_SERIAL
 KALEIDOSCOPE_EVENT_CONSUMER_SCADA
 KALEIDOSCOPE_EVENT_CONSUMER_USER1
 KALEIDOSCOPE_EVENT_CONSUMER_USER2
```

```
If block is non-zero, the call will block until a new event is posted or
a timeout occurs.  Otherwise, the call returns immediately.
```

Return Value

Returns 0 if non-blocking and no event. Returns 1 if an event is returned. Returns <0 on error.

Errors

No errors have yet been defined (always returns success).

Related information
SMART_KaleidoscopeCreate *(on page 32)*
SMART_KaleidoscopeAnalyzeFlush *(on page 30)*
SMART_KaleidoscopeAnalyzeGetZCFile *(on page 31)*
SMART_KaleidoscopeDelete *(on page 35)*
SMART_KaleidoscopeEventInit *(on page 36)*
SMART_KaleidoscopeEventPost *(on page 37)*
SMART_KaleidoscopeEventNext *(on page 36)*
SMART_GetSystemStatus *(on page 29)*
SMART_SetSystemStatus *(on page 42)*

SMART_KaleidoscopeEventPost

Post an event to the event logger

Synopsis

```
#include <smart.h>

int SMART_KaleidoscopeEventPost( const struct smart_kaleidoscope_results *event);
```

Description

Post the event to the event logger

Return Value

Returns 0 on success, -1 otherwise.

Errors

No errors have yet been defined (always returns success).

Related information
SMART_KaleidoscopeCreate *(on page 32)*
SMART_KaleidoscopeAnalyzeFlush *(on page 30)*
SMART_KaleidoscopeAnalyzeGetZCFile *(on page 31)*
SMART_KaleidoscopeDelete *(on page 35)*
SMART_KaleidoscopeEventInit *(on page 36)*
SMART_KaleidoscopeEventPost *(on page 37)*
SMART_KaleidoscopeEventNext *(on page 36)*
SMART_SetSystemStatus *(on page 42)*
SMART_GetSystemStatus *(on page 29)*

SMART_Open

open a connection to a SMART device such as a microphone.

Synopsis

```
#include <smart.h>

int SMART_Open(uint8_t * sn, uint8_t channelmask, uint8_t channel, uint32_t config, int *
 fdp);
```

Description

Given a sn 6 octet SMART device serial number (mac address), **SMART_Open()** returns a stream identifier, a small, nonnegative integer for use in subsequent calls **SMART_Read(), SMART_Write(),** and **SMART_Close().** Each stream is for a single audio channel specified by channel. If a SMART device is capable of multi-channel streaming (e.g. stereo, etc.), then each call to **SMART_Open()** must specify a channelmask indicating all of the channels to be opened on the SMART device. The first call to **SMART_Open()** will begin streaming all of the channels, but each individual call to **SMART_Open()** will attach to the specific channel identified by channel. The channel number is an integer beginning with zero, and the channelmask is a bitmask with channel 0 corresponding to the least-significant bit.

If fdp is non-NULL, **SMART_Open()** will write the underlying socket file descriptor to *fdp and mark the socket as non-blocking. In this way, callers can use select() to poll for events on the audio stream, but should still use **SMART_Read()** and **SMART_Write()** to read and write data from the stream using streamid.

The config is a bitmask of requested modes from the device and correspond to the capabilities of the device as defined in <smart_device_modes.h> to specify requested sample rate, high-pass filter setting, gains setting, sensor selection, calibration, and heating.

SMART_SR_MASK

Sample rates:

SMART_SR_8KHZ

8,000 Hz

SMART_SR_12KHZ

12,000 Hz

SMART_SR_16KHZ

16,000 Hz

SMART_SR_22KHZ

22,050 Hz

SMART_SR_24KHZ

24,000 Hz

SMART_SR_32KHZ

32,000 Hz

SMART_SR_44KHZ

44,100 Hz

SMART_SR_48KHZ

48,000 Hz

SMART_SR_96KHZ

96,000 Hz

SMART_SR_192KHZ

192,000 Hz

SMART_SR_384KHZ

384,000 Hz

SMART_SR_500KHZ

500,000 Hz

SMART_HP_MASK

High-pass filter settings:

SMART_HP_OFF

No high-pass filter set

SMART_HP_250HZ

250 Hz

SMART_HP_1KHZ

1 kHz

SMART_HP_8KHZ

8 kHz

SMART_HP_16KHZ

16 kHz

SMART_GAIN_MASK

Gain settings

SMART_GAIN_0DB

0 dB

SMART_GAIN_6DB

6 dB

SMART_GAIN_12DB

12 dB

SMART_GAIN_18DB

18 dB

SMART_GAIN_24DB

24 dB

SMART_GAIN_30DB

30 dB

SMART_GAIN_36DB

36 dB

SMART_GAIN_42DB

42 dB

SMART_CH_0

Select the primary microphone sensor

SMART_CH_1

Select the secondary (backup) microphone sensor

SMART_CH_CAL

Enable the calibrator (writes via **SMART_Write()** will send samples to the calibration transducer).

SMART_HEATER

Turn on the heating element (for defogging the sensors)

Return Value

SMART_Open() returns a stream id or -1 on error setting errno.

Errors

ECONNREFUSED

Unable to connect to the SMART daemon

EACCESS

Permission denied from missing or incorrect license, or unable to connect to SMART daemon socket.

ENODEV

Information about the requested device could not be found or there was a mismatch between the device capabilities and the requested configuration.

EINTR

Error occurred while communicating with SMART daemon. **EINVAL** Invalid configuraiton

Related information

smart-daemon *(on page 7)*
SMART_Read *(on page 40)*
SMART_Write *(on page 45)*
SMART_Close *(on page 23)*
SMART_GetDeviceInfo, SMART_GetDeviceInfoBySN, SMART_GetNDevices *(on page 25)*

SMART_Read

read audio samples from a SMART microphone device.

Synopsis

```
#include <smart.h>
#include <time.h>

int SMART_Read(int streamid, uint16_t * buffer, size_t length, struct timespec * tsp);
```

Description

Given a streamid returned from a previous call to **SMART_Open(),** read up to length bytes of 16-bit audio samples into buffer returning the number of bytes read.

If tsp is not NULL, a struct timespec will be written to *tsp indicating the time corresponding to the first sample in the returned buffer. The time has microsecond precision and is typically accurate to less than a millisecond.

The call is blocking unless opened with a non-null fdp specified in the previous call to **SMART_Open().**

Return Value

SMART_Read() returns the number of bytes written to buffer or -1 on error setting errno. A return value of zero indicates the SMART microphone device closed the connection.

Errors

EINVAL

Invalid streamid or an odd value for buflen.

EAGAIN or EWOULDBLOCK

The stream was opened non-blocking and no data is available.

Related information
SMART_Open *(on page 38)*
SMART_Write *(on page 45)*
SMART_Close *(on page 23)*
SMART_GetDeviceInfo, SMART_GetDeviceInfoBySN, SMART_GetNDevices *(on page 25)*

SMART_Reset

reset a SMART device such as a microphone

Synopsis

```
#include <smart.h>

int SMART_Reset(uint8_t *sn);
```

Description

The **SMART_Reset()** function sends a reset signal to the SMART device identified by a 6 octet serial number (mac address) by way of the SMART daemon

Return Value

SMART_Reset() return 0 on success or -1 on error setting errno.

Errors

ECONNREFUSED

Unable to connect to the SMART daemon

EACCESS

Permission denied from missing or incorrect license, or unable to connect to SMART daemon socket.

ENODEV

The requested device could not be found

EINTR

Error occurred while communicating with SMART daemon.

Related information
smart-daemon *(on page 7)*
SMART_Wake *(on page 44)*
SMART_Sleep *(on page 43)*
SMART_GetDeviceInfo, SMART_GetDeviceInfoBySN, SMART_GetNDevices *(on page 25)*

SMART_SetSystemStatus

Used by the logger service to update system status

Synopsis

```
#include <smart.h>

intSMART_SetSystemStatus();
```

Description

Used by the logger service to update the system status.

```
// System status
struct smart_system_status
  {
    time_t timestamp;// last update
    size_t bavail;   // available blocks
    size_t blocks;   // total blocks
    double voltage;  // voltage reading
    double load;     // load average
    size_t memtotal; // total memory (k)
    size_t memavail; // available memory (k)
  };
```

Return Value

Returns 0 on success, -1 otherwise.

Errors

No errors have yet been defined (always returns success).

Related information
SMART_KaleidoscopeCreate *(on page 32)*
SMART_KaleidoscopeAnalyzeFlush *(on page 30)*
SMART_KaleidoscopeAnalyzeGetZCFile *(on page 31)*
SMART_KaleidoscopeDelete *(on page 35)*
SMART_KaleidoscopeEventInit *(on page 36)*
SMART_KaleidoscopeEventPost *(on page 37)*
SMART_KaleidoscopeEventNext *(on page 36)*
SMART_GetSystemStatus *(on page 29)*
SMART_SetSystemStatus *(on page 42)*

SMART_Sleep

put a SMART device such as a microphone to sleep

Synopsis

```
#include <smart.h>

int SMART_Sleep(uint8_t *sn);
```

Description

The **SMART_Sleep()** function sends a sleep signal to the SMART device identified by a 6 octet serial number (mac address) by way of the SMART daemon to request that the device enter a low power sleep state.

Return Value

SMART_Sleep() return 0 on success or -1 on error setting errno.

Errors

ECONNREFUSED

Unable to connect to the SMART daemon

EACCESS

Permission denied from missing or incorrect license, or unable to connect to SMART daemon socket.

ENODEV

The requested device could not be found

EINTR

Error occurred while communicating with SMART daemon.

Related information

smart-daemon *(on page 7)*
SMART_Wake *(on page 44)*
SMART_Reset *(on page 41)*
SMART_GetDeviceInfo, SMART_GetDeviceInfoBySN, SMART_GetNDevices *(on page 25)*

SMART_Upgrade

upgrade a SMART device firmware

Synopsis

```
#include <smart.h>

int SMART_Upgrade(uint8_t *sn);
```

Description

The **SMART_Upgrade()** function sends an upgrade signal to the SMART device identified by a 6 octet serial number (mac address) by way of the SMART daemon to upgrade the device firmware. The SMART daemon will then initiate an upgrade sequence by sending the file at /usr/local/share/SMART/firmware_%d where %d is the device model number. This file would typically be a symbolic link to a model and version specific firmware

file. When the firmware upgrade is complete, the device should reboot and then run the new firmware. Applications should verify that the new firmware is running.

Return Value

SMART_Upgrade() return 0 on success or -1 on error setting errno.

Errors

ECONNREFUSED

Unable to connect to the SMART daemon

EACCESS

Permission denied from missing or incorrect license, or unable to connect to SMART daemon socket.

ENODEV

The requested device was not in a sleep state or could not be found.

EINTR

Error occurred while communicating with SMART daemon.

Related information
smart-daemon *(on page 7)*
SMART_Sleep *(on page 43)*
SMART_Reset *(on page 41)*
SMART_GetDeviceInfo, SMART_GetDeviceInfoBySN, SMART_GetNDevices *(on page 25)*

SMART_Wake

wake a SMART device that was previously sleeping

Synopsis

```
#include <smart.h>

int SMART_Wake(uint8_t *sn);
```

Description

The **SMART_Wake()** function sends a wake-up signal to the SMART device identified by a 6 octet serial number (mac address) by way of the SMART daemon to wake the device from sleep mode.

Return Value

SMART_Wake() return 0 on success or -1 on error setting errno.

Errors

ECONNREFUSED

Unable to connect to the SMART daemon

EACCESS

Permission denied from missing or incorrect license, or unable to connect to SMART daemon socket.

ENODEV

The requested device was not in a sleep state or could not be found.

EINTR

Error occurred while communicating with SMART daemon.

Related information
smart-daemon *(on page 7)*
SMART_Sleep *(on page 43)*
SMART_Reset *(on page 41)*
SMART_GetDeviceInfo, SMART_GetDeviceInfoBySN, SMART_GetNDevices *(on page 25)*

SMART_Write

write audio samples to a SMART device such as the calibratoion transducer of a microphone.

Synopsis

```
#include <smart.h>

int SMART_Write(int streamid, uint16_t * buffer, size_t length);
```

Description

Given a streamid returned from a previous call to **SMART_Open(),** write up to length bytes of 16-bit audio samples from buffer returning the number of bytes written.

The call is blocking unless opened with a non-null fdp specified in the previous call to **SMART_Open().**

Return Value

SMART_Write() returns the number of bytes written from buffer or -1 on error setting errno.

Errors

EINVAL

Invalid streamid or an odd value for buflen.

EAGAIN or EWOULDBLOCK

The stream was opened non-blocking and no data is available.

EPIPE

The connection was closed

Related information
SMART_Open *(on page 38)*
SMART_Read *(on page 40)*
SMART_Close *(on page 23)*
SMART_GetDeviceInfo, SMART_GetDeviceInfoBySN, SMART_GetNDevices *(on page 25)*

SMART_WriteWav

write audio samples to a .wav or .w4v file previously opened with **SMART_CreateWav().**

Synopsis

```
#include <smart.h>

int SMART_WriteWav(int handle, const uint16_t * buffer, size_t length);
```

Description

Given handle returned from a previous call to **SMART_CreateWav()** write length bytes of audio samples from buffer.

Return Value

SMART_WriteWav() returns the number of bytes written or -1 on error setting errno.

Errors

EINVAL

Invalid handle or length is ont even. **EIO** An error occurred trying to write data to the file.

Related information
SMART_CreateWav *(on page 24)*
SMART_CloseWav *(on page 23)*

Patents

The SMART System is covered under the following patents:

- AU 202210794
- AU 202210795
- AU 202210796
- AU 202210797
- GB 2559839
- GB 6188390
- GB 6188391
- GB 6188392
- GB 6188393
- EP 1661123
- EP 2877820
- US 8,995,230
- US 10,911,854

Copyright Notices

Contact Information

Wildlife Acoustics, Inc.
3 Mill and Main Place, Suite 110
Maynard, MA 01754
United States of America
+1 (978) 369-5225
U.S. toll-free: +1 (888) 733-0200
https://www.wildlifeacoustics.com