acoustics-hardware Documentation

Carl Andersson

Feb 17, 2021

Contents

This Python package provides a unified interface to serveral hardware platforms used in acoustic measurements, e.g. audio interfaces. The purpose of this package is twofold, both to enable cross-hardware interactions with a consistent interface, and to simplify scripting of advanced measurement setups.

Documentation: http://acoustics-hardware.readthedocs.io/

Source code and issue tracker: https://github.com/AppliedAcousticsChalmers/acoustics-hardware

CHAPTER 1

User API Documentation

1.1 Devices

class acoustics_hardware.core.**Device**(***kwargs*)

Abstract class that provides a consistent framework for different hardware.

An instance of a specific implementation of of *Device* is typically linked to a single physical input/output device. The instance manages connections to the device, any attached triggers, enabling/disabling input/output, and attached generators.

Variables

- **input_active** (Event) – Controls input state. Use set to activate input and clear to deactivate.
- **output_active** (Event) – Controls output state. Use set to activate output and clear to deactivate.
- **inputs** (list[*Channel*]) – List of assigned inputs, see *add_input*.
- **outputs** (list[*Channel*]) – List of assigned outputs, see *add_output*.
- **max_inputs** (int) – The maximum number of inputs available.
- **max_outputs** (int) – The maximum number of outputs available.
- **calibrations** (numpy.ndarray) – Calibrations of input channels, defaults to 1 for missing calibrations.

start()

Starts the device.

This creates the connections between the hardware and the software, configures the hardware, and initializes triggers and generators. Triggers are activated unless manually deactivated beforehand. Generators will not start generating data until the output is activated.

Note: This does NOT activate inputs or outputs!

stop()
Stops the device.

Use this to turn off or disconnect a device safely after a measurement. It is not recommended to use this as deactivation control, i.e. you should normally not have to make multiple calls to this function.

add_input(*index*, ***kwargs*)
Adds a new input *Channel*.

Parameters

- **index** (int) – Zero-based index of the channel.
- ****kwargs** – All arguments of *Channel* except chtype and index.

add_output(*index*, ***kwargs*)
Adds a new output *Channel*.

Parameters

- **index** (int) – Zero-based index of the channel.
- ****kwargs** – All arguments of *Channel* except chtype and index.

flush()
Used to flush all Qs.

This can be useful if a measurement needs to be discarded. Data that have been removed from the queues, e.g. automatic file writers, will not be interfered with.

Note: This will delete data which is still in the queues!

input_data()
Collects the acquired input data.

Data in stored internally in the *Device* object while input is active. This method is a convenient way to access the data from a measurement when more elaborate and automated setups are not required.

Returns numpy.ndarray – Array with the input data. Has the shape (n_inputs, n_samples), and the input channels are ordered in the same order as they were added.

calibrate(*channel*, *frequency=1000.0*, *value=1*, *ctype='rms'*, *unit='V'*)
Calibrates a channel using a reference signal.

The resulting calibration value and unit is stored as attributes of the corresponding *Channel*. Different calibration types should be used for different instruments. Currently only unfiltered RMS calibrations are implemented. This detects the level in the signal for 3 seconds, and uses the final level as the calibration value.

Parameters

- **channel** (int) – Index of the channel, in the order that they were added to the device.
- **frequency** (float) – The frequency of the applied reference signal, defaults to 1 kHz.
- **value** (float) – The value of the reference signal, defaults to 1.
- **ctype** ('rms') – Use to switch between different calibration methods. Currently not used.
- **unit** (str) – The unit of the calibrated quantity, defaults to 'V'.

add_distributor(*distributor*)
Adds a Distributor to the Device.

Parameters **distributor** – The distributor to add.

remove_distributor(*distributor*)
Removes a Distributor from the Device.

Parameters **distributor** – The distributor to remove.

add_trigger(*trigger*)
Adds a Trigger to the Device.

Parameters **trigger** – The trigger to add.

remove_trigger(*trigger*)
Removes a Trigger from the Device.

Parameters **trigger** – The trigger to remove.

add_generator(*generator*)
Adds a Generator to the Device.

Parameters **generator** – The generator to add.

Note: The order that multiple generators are added to a device dictates which output channel receives data from which generator. The total number of generated channels must match the number of output channels.

---

remove_generator(*generator*)
Removes a Generator from the Device.

Parameters **generator** – The generator to remove.

class acoustics_hardware.devices.**AudioDevice**(*name=None*, *fs=None*, *framesize=None*, ***kwargs*)
Class for interacting with audio interfaces.

Implementation of the *Device* framework for audio interfaces. Built on top of the sounddevice package.

Parameters

- **name** (str) – Partial or full name of the audio interface.
- **fs** (float, optional) – Sample rate for the device, defaults to system default for the device.
- **framesize** (int, optional) – The framesize for inputs and outputs, defaults to 1024 samples.

static get_devices(*name=None*)
Check which audio interfaces can be interacted with

Parameters **name** (str, optional) – incomplete name of device or None

Returns Complete name of device, or list of all devices.

class acoustics_hardware.devices.**NIDevice**(*name=None*, *fs=None*, *framesize=10000*, *dtype='float64'*, ***kwargs*)
Class for interacting with national instruments hardware.

Implementation of the *Device* framework for national instruments hardware. Built on top of the nidaqmx package.

Parameters

- **name** (str) – Partial or full name of the audio interface.

- **fs** (float, optional) – Sample rate for the device, defaults maximum supported rate.
- **framesize** (int, optional) – The framesize for inputs and outputs, defaults to 10000 samples.
- **dtype** (str, optional) – The datatype used while reading input data, default 'float64'.

static get_devices(*name=None*)
Check which NI hardware is available.

Since *NIDevice* is intended to interact with a single module at the time, this will complete names by selecting the first available module is a chassi.

Parameters **name** (str, optional) – incomplete name of device or None

Returns Complete name of device, or list of all devices.

input_range
Returns the input range on the device.

Note: This is only an approximate value, do NOT use for calibrating unscaled readings.

output_range
Returns the output range on the device.

Note: This is only an approximate value, do NOT use for calibrating unscaled outputs.

bit_depth(*channel=None*)
The bitdepth for the device.

Currently only implemented for input devices.

word_length(*channel=None*)
The word length for the device.

Only valid when using raw, unscaled, data types. Currently only implemented for input devices.

scaling_coeffs(*channels=None*)
Scaling coefficients used while reading raw input.

Returns the polynomial coefficients required to calculate input voltage from unscaled integers.

class acoustics_hardware.core.**Channel**(*index*, *chtype*, *label=None*, *calibration=None*, *unit=None*)
Represents a channel of a device.

Contains information about a physical channel used.

Parameters

- **index** (int) – Zero-based index of the channel in the device.
- **chtype** ('input' or 'output') – Type of channel.
- **label** (str, optional) – User label for identification of the channel.
- **calibration** (float, optional) – Manual calibration value.
- **unit** (str, optional) – Physical unit of the calibrated channel.

classmethod from_json(*json_dict*)
Creates a channel from json representation.

Parameters **json_dict** (`str`) – json representation of a dictionary containing key-value pairs for the arguments of a *Channel*.

Returns *Channel* – A channel with the given specification.

to_json()
Create json representation of this channel.

Returns `str` – json representation.

1.2 Triggers

class acoustics_hardware.triggers.**Trigger**(*action=None*, *false_action=None*, *auto_deactivate=True*, *use_calibrations=True*, *device=None*)
Base class for Trigger implementation.

A Trigger is an object that performs a test on all input data from a Device, regardless if the input is set as active or not. If the test evaluates to `True` the trigger will perform a set of actions, e.g. activate the input of a Device.

Parameters

- **actions** (*callable or list of callables*) – The actions that will be called each time the test evaluates to `True`.
- **false_actions** (*callable or list of callables*) – The actions that will be called each time the test evaluates to `False`.
- **auto_deactivate** (`bool`) – Sets if the trigger deactivates itself when the test is `True`. Useful to only trigger once, dafault `True`.
- **use_calibrations** (`bool`) – Sets if calibration values from the Device should be used for the test, default `True`.

Variables **active** (`Event`) – Controls if the trigger is active or not. A deactivated trigger will still test (e.g. to track levels), but not take action. Triggers start of as active unless manually deactivated.

test(*frame*)
Performs test.

The trigger conditions should be implemented here.

Parameters **frame** (`numpy.ndarray`) – The current input frame to test.

Returns `bool` – `True` -> do `actions`, `False` -> do `false_actions`

reset()
Resets the trigger state.

setup()
Configures trigger state.

class acoustics_hardware.triggers.**RMSTrigger**(*level*, *channel*, *region='Above'*, *level_detector_args=None*, ***kwargs*)
RMS level trigger.

Triggers actions based on a detected root-mean-square level.

Parameters

- **level** (`float`) – The level at which to trigger.

- **channel** (`int`) – The index of the channel on which to trigger.
- **region** (`'Above'` or `'Below'`, optional) – Defines if the triggering happens when the detected level rises above or falls below the set level, default `'Above'`.
- **level_detector_args** (`dict`, optional) – Passed as keyword arguments to the internal *LevelDetector*.
- ****kwargs** – Extra keyword arguments passed to *Trigger*.

class acoustics_hardware.triggers.**PeakTrigger**(*level*, *channel*, *region='Above'*, ***kwargs*)

Peak level trigger.

Triggers actions based on detected peak level.

Parameters

- **level** (`float`) – The level at which to trigger.
- **channel** (`int`) – The index of the channel on which to trigger.
- **region** (`'Above'` or `'Below'`, optional) – Defines if the triggering happens when the detected level rises above or falls below the set level, default `'Above'`.
- ****kwargs** – Extra keyword arguments passed to *Trigger*.

class acoustics_hardware.triggers.**DelayedAction**(*action*, *time*)

Delays an action.

When called, an instance of this class will excecute a specified action after a set delay. This can be useful to create timed measurements or pauses in a longer sequence.

Parameters

- **action** (*callable*) – Any callable action. This can be a callable class, a user defined funciton, or a method of another class. If several actions are required, create a lambda that calls all actions when called.
- **time** (`float`) – The delay time, in seconds.

1.3 Generators

class acoustics_hardware.generators.**Generator**(*device=None*, *amplitude=1*, ***kwargs*)

Base class for generator implementations.

A *Generator* is an object that creates data for output channels in a Device. Refer to specific generators for more details.

Variables **amplitude** (`float`) – The amplitude scale of the generator.

frame()

Generates a frame of output.

The generated frame must match the device framesize. If the generator creates multiple channels, it should have the shape `(n_ch, framesize)`, otherwise 1d arrays are sufficient.

Returns `numpy.ndarray` – Generated frame.

reset()

Resets the generator.

setup()
Configures the generator state.

exception acoustics_hardware.generators.**GeneratorStop**
Raised by Generators.

This exception indicates that the generator have reached some stopping criteria, e.g. end of file. Should be caught by the Device to stop output.

class acoustics_hardware.generators.**QGenerator**(***kwargs*)
Generator using queue.Queue.

Implementation of a *Generator* using a queue. Takes data from an input queue and generates frames with the correct framesize. The input queue must be filled fast enough otherwise the device output is cancelled.

Variables **Q** (Queue) – The queue from where data is extracted.

reset()
Clears the input queue.

class acoustics_hardware.generators.**ArbitrarySignalGenerator**(*repetitions=inf*, ***kwargs*)
Repeated generation of arbritrary signals.

Implementation of *Generator* for arbritrary signals.

Parameters

- **repetitions** (float) – The number of cycles to output before stopping, default np.inf.
- ****kwargs** – Will be saved as kwargs and accessible in *setup*.

Keyword Arguments **signal** (numpy.ndarray) – One cycle of the signal to output.

setup()
Configures the signal.

Create the signal manually and pass it while creating the generator as the signal argument.

It is possible to inherit *ArbitrarySignalGenerator* and override the setup method. Create one cycle of the signal and store it in self.signal. Access the underlying device as self.device, which has important properties, e.g. samplerate fs. All keyword arguments passed while creating instances are available as self.key.

Note: Call ArbitrarySignalGenerator.setup(self) from subclasses.

---

class acoustics_hardware.generators.**SweepGenerator**(*start_frequency*, *stop_frequency*, *duration*, *method='logarithmic'*, *bidirectional=False*, ***kwargs*)
Swept sine generator.

Parameters

- **start_frequency** (float) – Initial frequency of the sweep, in Hz.
- **stop_frequency** (float) – Final frequency of the sweep, in Hz.
- **duration** (float) – Duration of a single sweep, in seconds.
- **repetitions** (float, optional) – The number of repetitions, default np.inf.
- **method** (str, optional) – Chooses the type of sweep, see chirp, default 'logarithmic'.

- **bidirectional** (bool, optional) – If the sweep is bidirectional or not, default False.

See also:

ArbitrarySignalGenerator, scipy.signal.chirp

class acoustics_hardware.generators.**MaximumLengthSequenceGenerator**(*order*, ***kwargs*)

Generation of maximum length sequences.

Parameters

- **order** (int) – The order or the sequence. The total length is 2**order - 1.
- **repetitions** (float, optional) – The number of repetitions, default np.inf.

See also:

ArbitrarySignalGenerator, scipy.signal.max_len_seq

class acoustics_hardware.generators.**FunctionGenerator**(*frequency*, *repetitions=inf*, *shape='sine'*, *phase_offset=0*, *shape_kwargs=None*, ***kwargs*)

Generates signals from a shape function.

Implementation of *Generator* for standard funcitons.

Parameters

- **frequency** (float) – The frequecy of the signal, in Hz.
- **repetitions** (float, optional) – The number of repetitions, default np.inf.
- **shape** (str, optional) – Function shape, default 'sine'. Currently available functions are
  - 'sine': numpy.sin
  - 'sawtooth': scipy.signal.sawtooth
  - 'square': scipy.signal.square
- **phase_offset** (float, optional) – Phase offset of the signal in radians, default 0.
- **shape_kwargs** (dict) – Keyword arguments for shape function.

class acoustics_hardware.generators.**NoiseGenerator**(*color='white'*, *method='autoregressive'*, *ar_order=63*, ***kwargs*)

Generates colored noise.

Implementation of *Generator* for random noise signals.

Parameters

- **color** (str, optional) – The color of the noise. Each color corresponds to a inverse frequency power in the noise power density spectrum. Default is 'white'.
  - 'purple': -2
  - 'blue': -1
  - 'white': 0
  - 'pink': 1
  - 'brown': 2

- **method** (str) – The method used to create the noise. Currently two methods are implemented, a 'fft' method and an 'autoregressive' method. The default is 'autoregressive'. The autoregressive method is more expensive for small framesizes, but gives the same performance regardless of the framesize. The fft method have bad low-frequency performance for small framesizes.
- **ar_order** (int) – The order for the autoregressive method, default 63.

References

N. J. Kasdin, "Discrete simulation of colored noise and stochastic processes and 1/f^α power law noise generation," Proceedings of the IEEE, vol. 83, no. 5, pp. 802–827, May 1995. doi:10.1109/5.381848

1.4 Distributors

class acoustics_hardware.distributors.**Distributor**(*device=None*, ***kwargs*)
Base class for Distributors.

A *Distributor* is an object that should receive the input data from a Device, e.g. a plotter or a file writer. Refer to specific implementations for more details.

reset()
Resets the distributor

setup()
Configures the distributor state

class acoustics_hardware.distributors.**HDFWriter**(*filename=None*, ***kwargs*)
Implements writing to an HDF 5 file.

Hiearchial Data Format (HDF) 5 is a format suitable for multidimentional data. The format supports arbritrary metadata tags, and datasets can be organized in a folder-like structure within a single file.

Fileaccess is restricted to a single writer to maintain file integrity. A single writer can be be used to write data with multiple devices at the same time by using different datasets.

HDFWriter supports three different file writing modes, see *start*.

Parameters **name** (str) – The path to the file to write to.

Note: Only create a single HDFWriter per file!

---

add_input(*device=None*, *Q=None*)
Adds a new device or queue.

At least one of device or Q must be given. If device is None or a string the queue will be treated as an deviceless queue. If Q is not given it will be created and registred to the device.

Parameters

- **device** – A device which reads data.
- **Q** – A queue with data to write.

start(*mode='auto'*, *use_process=True*)
Starts the file writer.

The file writer will be started in one of three modes with different level of user controll.

Mode: `'auto'` The auto mode doc

Mode: `'signal'` The signal mode doc

Mode: `'manual'` The manual mode doc

Parameters

- **mode** (`str`) – The mode to start in.
- **use_process** (`bool`) – If writing should be managed in a separate process. This is recommended, and enabled by default, since file access with `h5py` will block all other threads. If set to false, file writing will instead be managed in a thread.

write(***kwargs*)

class acoustics_hardware.distributors.**HDFReader**(*filename*)
Implements reading from HDF 5 files

blocks(*start=None*, *stop=None*, *blocksize=None*)
A block generator. The last block might be of a different shape than the rest.

1.5 Processors

class acoustics_hardware.processors.**Processor**(*device=None*, ***kwargs*)
Base class for processors

A processor is an object that manipulates the data in some way.

process(*frame*)
Processes a single fraame of input.

The input frame might be the rame object as the read frame, so a processor should not manitulate the data in place.

Parameters **frame** (`numpy.ndarray`) – `(n_ch, n_samp)` shape input frame.

class acoustics_hardware.processors.**LevelDetector**(*channel*, *time_constant=0.05*, ***kwargs*)
Single channel level detector.

A level detector that tracks the root-mean-square level in a signal. The level is tracked with an exponentially decaying time average, implemented as a low-passed squared amplitude.

Parameters

- **channel** (`int`) – The index of the channel to track.
- **time_constant** (`float`) – Time constant for the exponential decay in seconds, default 50 ms.
- ****kwargs** – Extra arguments will be passed to *Processor*.

CHAPTER 2

genindex

Python Module Index

a

Index

M

N

O

P

Q

R

S

T

W