
waveline

Release 0.4.1

Lukas Berbuer (Vallen Systeme GmbH)

Jun 20, 2022

LIBRARY DOCUMENTATION

1	waveline.conditionwave	3
1.1	ConditionWave	3
1.2	Info	9
1.3	Setup	10
1.4	Status	12
2	waveline.spotwave	13
2.1	Info	13
2.2	Setup	14
2.3	SpotWave	15
2.4	Status	22
3	waveline.datatypes	23
3.1	AERecord	23
3.2	TRRecord	24
4	waveline.utils	27
4.1	decibel_to_volts	27
4.2	volts_to_decibel	27
5	Changelog	29
5.1	Unreleased	29
5.2	0.4.1 - 2022-06-20	29
5.3	0.4.0 - 2022-05-17	29
5.4	0.3.0 - 2021-06-15	30
5.5	0.2.0 - 2020-12-18	30
6	ToDoS	31
7	Indices and tables	33
	Python Module Index	35
	Index	37

Library to easily interface with Vallen Systeme WaveLine™ devices using the public APIs.

<i>waveline.conditionwave</i>	Module for conditionWave device.
<i>waveline.spotwave</i>	Module for spotWave device.
<i>waveline.datatypes</i>	Common datatypes.
<i>waveline.utils</i>	

WAVELINE.CONDITIONWAVE

Module for conditionWave device.

All device-related functions are exposed by the *ConditionWave* class.

Classes

<i>ConditionWave</i> (address)	Interface for conditionWave device.
<i>Info</i> (firmware_version, fpga_version, ...)	Device information.
<i>Setup</i> (adc_range_volts, adc_to_volts, ...)	Setup.
<i>Status</i> (temperature, buffer_size)	Status information.

1.1 ConditionWave

class waveline.conditionwave.*ConditionWave*(address)

Interface for conditionWave device.

The device is controlled via TCP/IP:

- Control port: 5432
- Streaming ports: 5433 for channel 1 and 5434 for channel 2

The interface is asynchronous and using *asyncio* for TCP/IP communication. This is especially beneficial for this kind of streaming applications, where most of the time the app is waiting for more data packets ([read more](#)). Please refer to the examples for implementation details.

__init__(address)

Initialize device.

Parameters

address (*str*) – IP address of device. Use the method *discover* to get IP addresses of available conditionWave devices.

Returns

Instance of *ConditionWave*

Example

There are two ways constructing and using the `ConditionWave` class:

1. Without context manager, manually calling the `connect` and `close` method:

```
>>> async def main():
>>>     cw = waveline.ConditionWave("192.168.0.100")
>>>     await cw.connect()
>>>     print(await cw.get_info())
>>>     ...
>>>     await cw.close()
>>> asyncio.run(main())
```

2. Using the async context manager:

```
>>> async def main():
>>>     async with waveline.ConditionWave("192.168.0.100") as cw:
>>>         print(await cw.get_info())
>>>         ...
>>> asyncio.run(main())
```

Methods

<code>__init__(address)</code>	Initialize device.
<code>acquire([raw])</code>	High-level method to continuously acquire data.
<code>close()</code>	Close connection.
<code>connect()</code>	Connect to device.
<code>discover([timeout])</code>	Discover conditionWave devices in network.
<code>get_ae_data()</code>	Get AE data records.
<code>get_info()</code>	Get device information.
<code>get_setup(channel)</code>	Get setup information.
<code>get_status()</code>	Get status information.
<code>get_tr_data([raw])</code>	Get transient data records.
<code>set_channel(channel, enabled)</code>	Enable/disable channel.
<code>set_continuous_mode(channel, enabled)</code>	Enable/disable continuous mode.
<code>set_ddt(channel, microseconds)</code>	Set duration discrimination time (DDT).
<code>set_filter(channel[, highpass, lowpass, order])</code>	Set IIR filter frequencies and order.
<code>set_range(channel, range_volts)</code>	Set input range.
<code>set_status_interval(channel, seconds)</code>	Set status interval.
<code>set_threshold(channel, microvolts)</code>	Set threshold for hit-based acquisition.
<code>set_tr_decimation(channel, factor)</code>	Set decimation factor of transient data and streaming data.
<code>set_tr_enabled(channel, enabled)</code>	Enable/disable recording of transient data.
<code>set_tr_postduration(channel, samples)</code>	Set post-duration samples for transient data.
<code>set_tr_pretrigger(channel, samples)</code>	Set pre-trigger samples for transient data.
<code>start_acquisition()</code>	Start data acquisition.
<code>start_pulsing(channel[, interval, count, cycles])</code>	Start pulsing.
<code>stop_acquisition()</code>	Stop data acquisition.
<code>stop_pulsing()</code>	Start pulsing.
<code>stream(channel, blocksize, *, raw)</code>	Async generator to stream channel data.

Attributes

<i>CHANNELS</i>	Available channels
<i>MAX_SAMPLERATE</i>	Maximum sampling rate in Hz
<i>PORT</i>	Control port number
<i>RANGES</i>	
<i>connected</i>	Check if connected to device.

CHANNELS = (1, 2)

Available channels

MAX_SAMPLERATE = 10000000

Maximum sampling rate in Hz

PORT = 5432

Control port number

classmethod discover(*timeout=0.5*)

Discover conditionWave devices in network.

Parameters

timeout (*float*) – Timeout in seconds

Return type

List[str]

Returns

List of IP addresses

property connected: *bool*

Check if connected to device.

Return type

bool

async connect()

Connect to device.

async close()

Close connection.

async get_info()

Get device information.

Return type

Info

async get_status()

Get status information.

Return type

Status

async get_setup(*channel*)

Get setup information.

Return type

Setup

async set_range(*channel*, *range_volts*)

Set input range.

Parameters

- **channel** (*int*) – Channel number (0 for all channels)
- **range_volts** (*float*) – Input range in volts (0.05, 5)

async set_channel(*channel*, *enabled*)

Enable/disable channel.

Parameters

- **channel** (*int*) – Channel number (0 for all channels)
- **enabled** (*bool*) – Set to *True* to enable channel

async set_continuous_mode(*channel*, *enabled*)

Enable/disable continuous mode.

Threshold will be ignored in continuous mode. The length of the records is determined by *ddt* with [set_ddt](#).

Parameters

- **channel** (*int*) – Channel number (0 for all channels)
- **enabled** (*bool*) – Set to *True* to enable continuous mode

async set_ddt(*channel*, *microseconds*)

Set duration discrimination time (DDT).

Parameters

- **channel** (*int*) – Channel number (0 for all channels)
- **microseconds** (*int*) – DDT in μ s

async set_status_interval(*channel*, *seconds*)

Set status interval.

Parameters

- **channel** (*int*) – Channel number (0 for all channels)
- **seconds** (*int*) – Status interval in s

async set_tr_enabled(*channel*, *enabled*)

Enable/disable recording of transient data.

Parameters

- **channel** (*int*) – Channel number (0 for all channels)
- **enabled** (*bool*) – Set to *True* to enable transient data

async set_tr_decimation(*channel*, *factor*)

Set decimation factor of transient data and streaming data.

The sampling rate will be 10 MHz / *factor*.

Parameters

- **channel** (*int*) – Channel number (0 for all channels)
- **factor** (*int*) – Decimation factor

async set_tr_pretrigger(*channel, samples*)

Set pre-trigger samples for transient data.

Parameters

- **channel** (*int*) – Channel number (0 for all channels)
- **samples** (*int*) – Pre-trigger samples

async set_tr_postduration(*channel, samples*)

Set post-duration samples for transient data.

Parameters

- **channel** (*int*) – Channel number (0 for all channels)
- **samples** (*int*) – Post-duration samples

async set_filter(*channel, highpass=None, lowpass=None, order=8*)

Set IIR filter frequencies and order.

Parameters

- **channel** (*int*) – Channel number (0 for all channels)
- **highpass** (*Optional[float]*) – Highpass frequency in Hz (*None* to disable highpass filter)
- **lowpass** (*Optional[float]*) – Lowpass frequency in Hz (*None* to disable lowpass filter)
- **order** (*int*) – Filter order

async set_threshold(*channel, microvolts*)

Set threshold for hit-based acquisition.

Parameters

- **channel** (*int*) – Channel number (0 for all channels)
- **microvolts** (*float*) – Threshold in μV

async start_acquisition()

Start data acquisition.

async stop_acquisition()

Stop data acquisition.

async start_pulsing(*channel, interval=1, count=4, cycles=1*)

Start pulsing.

The number of pulses should be even, because pulses are generated by a square-wave signal (between LOW and HIGH) and the pulse signal should end LOW.

Parameters

- **channel** (*int*) – Channel number (0 for all channels)
- **interval** (*float*) – Interval between pulses in seconds
- **count** (*int*) – Number of pulses per channel (should be even), 0 for infinite pulses
- **cycles** (*int*) – Number of pulse cycles (automatically pulse through each channel in cycles). Only useful if all channels are chosen.

async stop_pulsing()

Start pulsing.

Args:

async get_ae_data()

Get AE data records.

Return type

`List[AERecord]`

Returns

List of AE data records (either status or hit data)

async get_tr_data(raw=False)

Get transient data records.

Parameters

raw (`bool`) – Return TR amplitudes as ADC values if *True*, skip conversion to volts

Return type

`List[TRRecord]`

Returns

List of transient data records

async acquire(raw=False)

High-level method to continuously acquire data.

Parameters

raw (`bool`) – Return TR amplitudes as ADC values if *True*, skip conversion to volts

Yields

AE and TR data records

Example

```
>>> async with waveline.ConditionWave("192.254.100.100") as cw:
>>>     # apply settings
>>>     await cw.set_channel(channel=1, enabled=True)
>>>     await cw.set_channel(channel=2, enabled=False)
>>>     await cw.set_range(channel=1, range_volts=0.05)
>>>     async for record in cw.acquire():
>>>         # do something with the data depending on the type
>>>         if isinstance(record, waveline.AERecord):
>>>             ...
>>>         if isinstance(record, waveline.TRRecord):
>>>             ...
```

Return type

`AsyncIterator[Union[AERecord, TRRecord]]`

stream(channel, blocksize, *, raw=False)

Async generator to stream channel data.

Parameters

- **channel** (`int`) – Channel number [1, 2]

- **blocksize** (`int`) – Number of samples per block
- **raw** (`bool`) – Return ADC values if *True*, skip conversion to volts

Yields

Tuple of

- relative time in seconds (first block: $t = 0$)
- data as numpy array in volts (or ADC values if *raw* is *True*)

Example

```
>>> async with waveline.ConditionWave("192.168.0.100") as cw:
>>>     # apply settings
>>>     await cw.set_range(0.05)
>>>     await cw.set_filter(100e3, 500e3, 8)
>>>     # open streaming port before start acq afterwards (order matters!)
>>>     stream = cw.stream(channel=1, blocksize=65536)
>>>     await cw.start_acquisition()
>>>     async for time, block in stream:
>>>         # do something with the data
>>>         ...
```

Return type

`AsyncIterator[Tuple[float, ndarray]]`

1.2 Info

class `waveline.conditionwave.Info`(*firmware_version*, *fpga_version*, *channel_count*, *range_count*, *max_sample_rate*, *adc_to_volts*)

Device information.

__init__(*firmware_version*, *fpga_version*, *channel_count*, *range_count*, *max_sample_rate*, *adc_to_volts*)

Methods

__init__(*firmware_version*, *fpga_version*, ...)

Attributes

<code>firmware_version</code>	Firmware version
<code>fpga_version</code>	FPGA version
<code>channel_count</code>	Number of channels
<code>range_count</code>	Number of selectable ranges
<code>max_sample_rate</code>	Max sampling rate
<code>adc_to_volts</code>	Conversion factors from ADC values to V for both ranges

firmware_version: `str`
Firmware version

fpga_version: `str`
FPGA version

channel_count: `int`
Number of channels

range_count: `int`
Number of selectable ranges

max_sample_rate: `float`
Max sampling rate

adc_to_volts: `List[float]`
Conversion factors from ADC values to V for both ranges

1.3 Setup

```
class waveline.conditionwave.Setup(adc_range_volts, adc_to_volts, filter_highpass_hz, filter_lowpass_hz,  
filter_order, enabled, continuous_mode, threshold_volts, ddt_seconds,  
status_interval_seconds, tr_enabled, tr_decimation,  
tr_pretrigger_samples, tr_postduration_samples)
```

Setup.

```
__init__(adc_range_volts, adc_to_volts, filter_highpass_hz, filter_lowpass_hz, filter_order, enabled,  
continuous_mode, threshold_volts, ddt_seconds, status_interval_seconds, tr_enabled,  
tr_decimation, tr_pretrigger_samples, tr_postduration_samples)
```

Methods

```
__init__(adc_range_volts, adc_to_volts, ...)
```

Attributes

<i>adc_range_volts</i>	ADC input range in volts
<i>adc_to_volts</i>	Conversion factor from ADC values to volts
<i>filter_highpass_hz</i>	Highpass frequency in Hz
<i>filter_lowpass_hz</i>	Lowpass frequency in Hz
<i>filter_order</i>	Filter order
<i>enabled</i>	Flag if channel is enabled
<i>continuous_mode</i>	Flag if continuous mode is enabled
<i>threshold_volts</i>	Threshold for hit-based acquisition in volts
<i>ddt_seconds</i>	Duration discrimination time (DDT) in seconds
<i>status_interval_seconds</i>	Status interval in seconds
<i>tr_enabled</i>	Flag in transient data recording is enabled
<i>tr_decimation</i>	Decimation factor for transient data
<i>tr_pretrigger_samples</i>	Pre-trigger samples for transient data
<i>tr_postduration_samples</i>	Post-duration samples for transient data

adc_range_volts: `float`

ADC input range in volts

adc_to_volts: `float`

Conversion factor from ADC values to volts

filter_highpass_hz: `Optional[float]`

Highpass frequency in Hz

filter_lowpass_hz: `Optional[float]`

Lowpass frequency in Hz

filter_order: `int`

Filter order

enabled: `bool`

Flag if channel is enabled

continuous_mode: `bool`

Flag if continuous mode is enabled

threshold_volts: `float`

Threshold for hit-based acquisition in volts

ddt_seconds: `float`

Duration discrimination time (DDT) in seconds

status_interval_seconds: `float`

Status interval in seconds

tr_enabled: `bool`

Flag in transient data recording is enabled

tr_decimation: `int`

Decimation factor for transient data

tr_pretrigger_samples: `int`

Pre-trigger samples for transient data

tr_postduration_samples: `int`

Post-duration samples for transient data

1.4 Status

class waveline.conditionwave.**Status**(*temperature*, *buffer_size*)

Status information.

__init__(*temperature*, *buffer_size*)

Methods

`__init__`(*temperature*, *buffer_size*)

Attributes

<i>temperature</i>	Device temperature in °C
<i>buffer_size</i>	Buffer size in bytes

temperature: `float`

Device temperature in °C

buffer_size: `int`

Buffer size in bytes

WAVELINE.SPOTWAVE

Module for spotWave device.

All device-related functions are exposed by the *SpotWave* class.

Classes

<i>Info</i> (firmware_version, type_, model, ...)	Device information.
<i>Setup</i> (recording, logging, cont_enabled, ...)	Setup.
<i>SpotWave</i> (port)	Interface for spotWave devices.
<i>Status</i> (temperature, recording, logging, ...)	Status information.

2.1 Info

class waveline.spotwave.**Info**(firmware_version, type_, model, input_range_decibel)

Device information.

__init__(firmware_version, type_, model, input_range_decibel)

Methods

__init__(firmware_version, type_, model, ...)

Attributes

<i>firmware_version</i>	Firmware version (major, minor)
<i>type_</i>	Device type
<i>model</i>	Model identifier
<i>input_range_decibel</i>	Input range in dBAE

firmware_version: **str**

Firmware version (major, minor)

type_: `str`
Device type

model: `str`
Model identifier

input_range_decibel: `int`
Input range in dBAE

2.2 Setup

```
class waveline.spotwave.Setup(recording, logging, cont_enabled, adc_to_volts, threshold_volts, ddt_seconds,
                               status_interval_seconds, filter_highpass_hz, filter_lowpass_hz, filter_order,
                               tr_enabled, tr_decimation, tr_pretrigger_samples, tr_postduration_samples,
                               cct_seconds)
```

Setup.

```
__init__(recording, logging, cont_enabled, adc_to_volts, threshold_volts, ddt_seconds,
          status_interval_seconds, filter_highpass_hz, filter_lowpass_hz, filter_order, tr_enabled,
          tr_decimation, tr_pretrigger_samples, tr_postduration_samples, cct_seconds)
```

Methods

```
__init__(recording, logging, cont_enabled, ...)
```

Attributes

<code>recording</code>	Flag if acquisition is active
<code>logging</code>	Flag if logging is active
<code>cont_enabled</code>	Flag if continuous mode is enabled
<code>adc_to_volts</code>	Conversion factor from ADC values to volts
<code>threshold_volts</code>	Threshold for hit-based acquisition in volts
<code>ddt_seconds</code>	Duration discrimination time (DDT) in seconds
<code>status_interval_seconds</code>	Status interval in seconds
<code>filter_highpass_hz</code>	Highpass frequency in Hz
<code>filter_lowpass_hz</code>	Lowpass frequency in Hz
<code>filter_order</code>	Filter order
<code>tr_enabled</code>	Flag in transient data recording is enabled
<code>tr_decimation</code>	Decimation factor for transient data
<code>tr_pretrigger_samples</code>	Pre-trigger samples for transient data
<code>tr_postduration_samples</code>	Post-duration samples for transient data
<code>cct_seconds</code>	Coupling check transmitter (CCT) / pulser interval in seconds

recording: `bool`
Flag if acquisition is active

logging: `bool`
Flag if logging is active

cont_enabled: `bool`
Flag if continuous mode is enabled

adc_to_volts: `float`
Conversion factor from ADC values to volts

threshold_volts: `float`
Threshold for hit-based acquisition in volts

ddt_seconds: `float`
Duration discrimination time (DDT) in seconds

status_interval_seconds: `float`
Status interval in seconds

filter_highpass_hz: `Optional[float]`
Highpass frequency in Hz

filter_lowpass_hz: `Optional[float]`
Lowpass frequency in Hz

filter_order: `int`
Filter order

tr_enabled: `bool`
Flag in transient data recording is enabled

tr_decimation: `int`
Decimation factor for transient data

tr_pretrigger_samples: `int`
Pre-trigger samples for transient data

tr_postduration_samples: `int`
Post-duration samples for transient data

cct_seconds: `float`
Coupling check transmitter (CCT) / pulser interval in seconds

2.3 SpotWave

class `waveline.spotwave.SpotWave(port)`

Interface for spotWave devices.

The spotWave device is connected via USB and exposes a virtual serial port for communication.

__init__(*port*)

Initialize device.

Parameters

port (`Union[str, Serial]`) – Either the serial port id (e.g. “COM6”) or a `serial.Serial` port instance. Use the method [discover](#) to get a list of ports with connected spotWave devices.

Returns

Instance of *SpotWave*

Example

There are two ways constructing and using the *SpotWave* class:

1. Without context manager and manually calling the *close* method afterwards:

```
>>> sw = waveline.SpotWave("COM6")
>>> print(sw.get_setup())
>>> ...
>>> sw.close()
```

2. Using the context manager:

```
>>> with waveline.SpotWave("COM6") as sw:
>>>     print(sw.get_setup())
>>>     ...
```

Methods

<code>__init__(port)</code>	Initialize device.
<code>acquire([raw])</code>	High-level method to continuously acquire data.
<code>clear_buffer()</code>	Clear input and output buffer.
<code>clear_data_log()</code>	Clear logged data from internal memory.
<code>close()</code>	Close serial connection to the device.
<code>connect()</code>	Open serial connection to the device.
<code>discover()</code>	Discover connected spotWave devices.
<code>get_ae_data()</code>	Get AE data records.
<code>get_data(samples[, raw])</code>	Read snapshot of transient data with maximum sampling rate (2 MHz).
<code>get_data_log()</code>	Get logged AE data records data from internal memory
<code>get_info()</code>	Get device information.
<code>get_setup()</code>	Get setup.
<code>get_status()</code>	Get status.
<code>get_tr_data([raw])</code>	Get transient data records.
<code>readlines()</code>	
<code>set_cct(interval_seconds[, sync])</code>	Set coupling check ransmitter (CCT) / pulser interval.
<code>set_continuous_mode(enabled)</code>	Enable/disable continuous mode.
<code>set_datetime([timestamp])</code>	Set current date and time.
<code>set_ddt(microseconds)</code>	Set duration discrimination time (DDT).
<code>set_filter([highpass, lowpass, order])</code>	Set IIR filter frequencies and order.
<code>set_logging_mode(enabled)</code>	Enable/disable data log mode.
<code>set_status_interval(seconds)</code>	Set status interval.
<code>set_threshold(microvolts)</code>	Set threshold for hit-based acquisition.
<code>set_tr_decimation(factor)</code>	Set decimation factor of transient data.
<code>set_tr_enabled(enabled)</code>	Enable/disable recording of transient data.
<code>set_tr_postduration(samples)</code>	Set post-duration samples for transient data.
<code>set_tr_pretrigger(samples)</code>	Set pre-trigger samples for transient data.
<code>start_acquisition()</code>	Start acquisition.
<code>stop_acquisition()</code>	Stop acquisition.
<code>stream(*args, **kwargs)</code>	Alias for <i>SpotWave.acquire</i> method.

Attributes

<code>CLOCK</code>	Internal clock in Hz
<code>PRODUCT_ID</code>	USB product id of SpotWave device
<code>VENDOR_ID</code>	USB vendor id of Vallen Systeme GmbH
<code>connected</code>	Check if the connection to the device is open.

VENDOR_ID = 8849

USB vendor id of Vallen Systeme GmbH

PRODUCT_ID = 272

USB product id of SpotWave device

CLOCK = 2000000

Internal clock in Hz

connect()

Open serial connection to the device.

The `connect` method is automatically called in the constructor. You only need to call the method to reopen the connection after calling `close`.

close()

Close serial connection to the device.

property connected: bool

Check if the connection to the device is open.

Return type

`bool`

classmethod discover()

Discover connected spotWave devices.

Return type

`List[str]`

Returns

List of port names

clear_buffer()

Clear input and output buffer.

get_info()

Get device information.

Return type

`Info`

Returns

Dataclass with device information

get_setup()

Get setup.

Return type

`Setup`

Returns

Dataclass with setup information

get_status()

Get status.

Return type

`Status`

Returns

Dataclass with status information

set_continuous_mode(enabled)

Enable/disable continuous mode.

Threshold will be ignored in continuous mode. The length of the records is determined by `ddt` with `set_ddt`.

Note: The parameters for continuous mode with transient recording enabled (`set_tr_enabled`) have to be chosen with care - mainly the decimation factor (`set_tr_decimation`) and `ddt` (`set_ddt`). The internal buffer of the device can store up to ~200.000 samples.

If the buffer is full, data records are lost. Small latencies in data polling can cause overflows and therefore data loss. One record should not exceed half the buffer size (~100.000 samples). 25% of the buffer size (~50.000 samples) is a good starting point. The number of samples in a record is determined by `ddt` and the decimation factor `d`: $n = ddt_{\mu s} \cdot f_s / d = ddt_{\mu s} \cdot 2 / d \implies ddt_{\mu s} \approx 50.000 \cdot d / 2$

On the other hand, if the number of samples is small, more hits are generated and the CPU load increases.

Parameters

enabled (`bool`) – Set to *True* to enable continuous mode

set_ddt(*microseconds*)

Set duration discrimination time (DDT).

Parameters

microseconds (`int`) – DDT in μs

set_status_interval(*seconds*)

Set status interval.

Parameters

seconds (`int`) – Status interval in s

set_tr_enabled(*enabled*)

Enable/disable recording of transient data.

Parameters

enabled (`bool`) – Set to *True* to enable transient data

set_tr_decimation(*factor*)

Set decimation factor of transient data.

The sampling rate of transient data will be 2 MHz / *factor*.

Parameters

factor (`int`) – Decimation factor

set_tr_pretrigger(*samples*)

Set pre-trigger samples for transient data.

Parameters

samples (`int`) – Pre-trigger samples

set_tr_postduration(*samples*)

Set post-duration samples for transient data.

Parameters

samples (`int`) – Post-duration samples

set_cct(*interval_seconds*, *sync=False*)

Set coupling check transmitter (CCT) / pulser interval.

The pulser amplitude is 3.3 V.

Parameters

- **interval_seconds** (*int*) – Pulser interval in seconds
- **sync** (*bool*) – Synchronize the pulser with the first sample of the *get_data* command

set_filter(*highpass=None, lowpass=None, order=4*)

Set IIR filter frequencies and order.

Parameters

- **highpass** (*Optional[float]*) – Highpass frequency in Hz (*None* to disable highpass filter)
- **lowpass** (*Optional[float]*) – Lowpass frequency in Hz (*None* to disable lowpass filter)
- **order** (*int*) – Filter order

set_datetime(*timestamp=None*)

Set current date and time.

Parameters

timestamp (*Optional[datetime]*) – *datetime.datetime* object, current time if *None*

set_threshold(*microvolts*)

Set threshold for hit-based acquisition.

Parameters

microvolts (*float*) – Threshold in μV

set_logging_mode(*enabled*)

Enable/disable data log mode.

Parameters

enabled (*bool*) – Set to *True* to enable logging mode

start_acquisition()

Start acquisition.

stop_acquisition()

Stop acquisition.

get_ae_data()

Get AE data records.

Todo:

- Implement parsing of record start marker
-

Return type

List[AERecord]

Returns

List of AE data records (either status or hit data)

get_tr_data(*raw=False*)

Get transient data records.

Parameters

raw (*bool*) – Return TR amplitudes as ADC values if *True*, skip conversion to volts

Return type

`List[TRRecord]`

Returns

List of transient data records

acquire(*raw=False*)

High-level method to continuously acquire data.

Parameters

raw (`bool`) – Return TR amplitudes as ADC values if *True*, skip conversion to volts

Yields

AE and TR data records

Example

```
>>> with waveline.SpotWave("COM6") as sw:
>>>     # apply settings
>>>     sw.set_ddt(400)
>>>     for record in sw.stream():
>>>         # do something with the data depending on the type
>>>         if isinstance(record, waveline.AERecord):
>>>             ...
>>>         if isinstance(record, waveline.TRRecord):
>>>             ...
```

Return type

`Iterator[Union[AERecord, TRRecord]]`

stream(*args, **kwargs)

Alias for *SpotWave.acquire* method.

Deprecated: Please use the *acquire* method instead.

get_data(*samples*, *raw=False*)

Read snapshot of transient data with maximum sampling rate (2 MHz).

Parameters

- **samples** (`int`) – Number of samples to read
- **raw** (`bool`) – Return ADC values if *True*, skip conversion to volts

Return type

`ndarray`

Returns

Array with amplitudes in volts (or ADC values if *raw* is *True*)

get_data_log()

Get logged AE data records data from internal memory

Return type

`List[AERecord]`

Returns

List of AE data records (either status or hit data)

clear_data_log()

Clear logged data from internal memory.

2.4 Status

class waveline.spotwave.**Status**(*temperature, recording, logging, log_data_usage, datetime*)

Status information.

__init__(*temperature, recording, logging, log_data_usage, datetime*)

Methods

__init__(*temperature, recording, logging, ...*)

Attributes

<i>temperature</i>	Device temperature in °C
<i>recording</i>	Flag if acquisition is active
<i>logging</i>	Flag if logging is active
<i>log_data_usage</i>	Log buffer usage in sets
<i>datetime</i>	Device datetime

temperature: **int**

Device temperature in °C

recording: **bool**

Flag if acquisition is active

logging: **bool**

Flag if logging is active

log_data_usage: **int**

Log buffer usage in sets

datetime: **datetime**

Device datetime

WAVELINE.DATATYPES

Common datatypes.

Classes

<code><i>AERecord</i>(type_, channel, time, amplitude, ...)</code>	AE data record, either status or hit data.
<code><i>TRRecord</i>(channel, trai, time, samples, data)</code>	Transient data record.

3.1 AERecord

class `waveline.datatypes.AERecord`(*type_*, *channel*, *time*, *amplitude*, *rise_time*, *duration*, *counts*, *energy*, *trai*, *flags*)

AE data record, either status or hit data.

Todo:

- Documentation or data type with available hit flags
-

`__init__`(*type_*, *channel*, *time*, *amplitude*, *rise_time*, *duration*, *counts*, *energy*, *trai*, *flags*)

Methods

`__init__`(*type_*, *channel*, *time*, *amplitude*, ...)

Attributes

<code>type_</code>	Record type (hit or status data)
<code>channel</code>	Channel number
<code>time</code>	Time in seconds
<code>amplitude</code>	Peak amplitude in volts
<code>rise_time</code>	Rise time in seconds
<code>duration</code>	Duration in seconds
<code>counts</code>	Number of positive threshold crossings
<code>energy</code>	Energy (EN 1330-9) in eu (1e-14 V ² s)
<code>trai</code>	Transient recorder index (key between AERecord and TRRecord)
<code>flags</code>	Hit flags

type_: `str`
Record type (hit or status data)

channel: `int`
Channel number

time: `float`
Time in seconds

amplitude: `float`
Peak amplitude in volts

rise_time: `float`
Rise time in seconds

duration: `float`
Duration in seconds

counts: `int`
Number of positive threshold crossings

energy: `float`
Energy (EN 1330-9) in eu (1e-14 V²s)

trai: `int`
Transient recorder index (key between [AERecord](#) and [TRRecord](#))

flags: `int`
Hit flags

3.2 TRRecord

```
class waveline.datatypes.TRRecord(channel, trai, time, samples, data, raw=False)
    Transient data record.
    __init__(channel, trai, time, samples, data, raw=False)
```

Methods

`__init__(channel, trai, time, samples, data)`

Attributes

<code>raw</code>	ADC values instead of user values (volts)
<code>channel</code>	Channel number
<code>trai</code>	Transient recorder index (key between <i>AERecord</i> and <i>TRRecord</i>)
<code>time</code>	Time in seconds
<code>samples</code>	Number of samples
<code>data</code>	Array of transient data in volts (or ADC values if <code>raw</code> is <i>True</i>)

channel: `int`

Channel number

trai: `int`

Transient recorder index (key between *AERecord* and *TRRecord*)

time: `float`

Time in seconds

samples: `int`

Number of samples

data: `ndarray`

Array of transient data in volts (or ADC values if `raw` is *True*)

raw: `bool = False`

ADC values instead of user values (volts)

WAVELINE.UTILS

Functions

<code>decibel_to_volts(decibel)</code>	Convert from dB(AE) to volts.
<code>volts_to_decibel(volts)</code>	Convert from volts to dB(AE).

4.1 decibel_to_volts

`waveline.utils.decibel_to_volts(decibel)`

Convert from dB(AE) to volts.

Parameters

decibel (`Union[float, ndarray]`) – Input in decibel, scalar or array

Return type

`Union[float, ndarray]`

Returns

Input value(s) in volts

4.2 volts_to_decibel

`waveline.utils.volts_to_decibel(volts)`

Convert from volts to dB(AE).

Parameters

volts (`Union[float, ndarray]`) – Input in volts, scalar or array

Return type

`Union[float, ndarray]`

Returns

Input value(s) in dB(AE)

CHANGELOG

All notable changes to this project will be documented in this file.

The format is based on [Keep a Changelog](#), and this project adheres to [Semantic Versioning](#).

5.1 Unreleased

5.2 0.4.1 - 2022-06-20

5.2.1 Fixed

- [conditionWave] Increase TCP read timeout for `get_ae_data` / `get_tr_data` to prevent timeout errors

5.3 0.4.0 - 2022-05-17

5.3.1 Added

- [conditionWave] Add all commands of new firmware (hit-based acquisition, pulsing, ...)
- [conditionWave] Add example for hit-based acquisition
- Add Python 3.9 and 3.10 to CI pipeline

5.3.2 Changed

- [conditionWave] Rename `set_decimation` method to `set_tr_decimation`
- [conditionWave] Remove `get_temperature` and `get_buffersize` method (replace with `get_status` method)
- [spotWave] Rename `stream` method to `acquire`. `stream` method is still an alias but deprecated and will be removed in the future

5.3.3 Fixed

- [conditionWave] Wait for all stream connection before `start_acquisition`

5.4 0.3.0 - 2021-06-15

5.4.1 Added

- [conditionWave] Multi-channel example
- [conditionWave] Optional `start` argument (timestamp) for `stream`
- [spotWave] Add examples
- [spotWave] Add firmware check

5.4.2 Changed

- [conditionWave] Channel arguments for `set_range`, `set_decimation` and `set_filter`
- [spotWave] `set_status_interval` with seconds instead of milliseconds
- [spotWave] Require firmware ≥ 00.25

5.4.3 Removed

- [conditionWave] Properties `input_range`, `decimation`, `filter_settings`

5.4.4 Fixed

- [conditionWave] ADC to volts conversion factor
- [spotWave] Aggregate TR/AE records to prevent IO timeouts

5.5 0.2.0 - 2020-12-18

First public release

TODOS

Todo:

- Documentation or data type with available hit flags
-

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/pywaveline/envs/0.4.1/lib/python3.7/site-packages/waveline/datatypes.py:docstring of waveline.datatypes.AERecord, line 3.`)

Todo:

- Implement parsing of record start marker
-

(The [original entry](#) is located in `/home/docs/checkouts/readthedocs.org/user_builds/pywaveline/envs/0.4.1/lib/python3.7/site-packages/waveline/spotwave.py:docstring of waveline.spotwave.SpotWave.get_ae_data, line 3.`)

INDICES AND TABLES

- `genindex`
- `modindex`

PYTHON MODULE INDEX

W

`waveline.conditionwave`, [3](#)

`waveline.datatypes`, [23](#)

`waveline.spotwave`, [13](#)

`waveline.utils`, [27](#)

Symbols

`__init__()` (*waveline.conditionwave.ConditionWave method*), 3
`__init__()` (*waveline.conditionwave.Info method*), 9
`__init__()` (*waveline.conditionwave.Setup method*), 10
`__init__()` (*waveline.conditionwave.Status method*), 12
`__init__()` (*waveline.datatypes.AERRecord method*), 23
`__init__()` (*waveline.datatypes.TRRecord method*), 24
`__init__()` (*waveline.spotwave.Info method*), 13
`__init__()` (*waveline.spotwave.Setup method*), 14
`__init__()` (*waveline.spotwave.SpotWave method*), 15
`__init__()` (*waveline.spotwave.Status method*), 22

A

`acquire()` (*waveline.conditionwave.ConditionWave method*), 8
`acquire()` (*waveline.spotwave.SpotWave method*), 21
`adc_range_volts` (*waveline.conditionwave.Setup attribute*), 11
`adc_to_volts` (*waveline.conditionwave.Info attribute*), 10
`adc_to_volts` (*waveline.conditionwave.Setup attribute*), 11
`adc_to_volts` (*waveline.spotwave.Setup attribute*), 15
`AERRecord` (*class in waveline.datatypes*), 23
`amplitude` (*waveline.datatypes.AERRecord attribute*), 24

B

`buffer_size` (*waveline.conditionwave.Status attribute*), 12

C

`cct_seconds` (*waveline.spotwave.Setup attribute*), 15
`channel` (*waveline.datatypes.AERRecord attribute*), 24
`channel` (*waveline.datatypes.TRRecord attribute*), 25
`channel_count` (*waveline.conditionwave.Info attribute*), 10
`CHANNELS` (*waveline.conditionwave.ConditionWave attribute*), 5
`clear_buffer()` (*waveline.spotwave.SpotWave method*), 18

`clear_data_log()` (*waveline.spotwave.SpotWave method*), 21
`CLOCK` (*waveline.spotwave.SpotWave attribute*), 17
`close()` (*waveline.conditionwave.ConditionWave method*), 5
`close()` (*waveline.spotwave.SpotWave method*), 18
`ConditionWave` (*class in waveline.conditionwave*), 3
`connect()` (*waveline.conditionwave.ConditionWave method*), 5
`connect()` (*waveline.spotwave.SpotWave method*), 17
`connected` (*waveline.conditionwave.ConditionWave property*), 5
`connected` (*waveline.spotwave.SpotWave property*), 18
`cont_enabled` (*waveline.spotwave.Setup attribute*), 15
`continuous_mode` (*waveline.conditionwave.Setup attribute*), 11
`counts` (*waveline.datatypes.AERRecord attribute*), 24

D

`data` (*waveline.datatypes.TRRecord attribute*), 25
`datetime` (*waveline.spotwave.Status attribute*), 22
`ddt_seconds` (*waveline.conditionwave.Setup attribute*), 11
`ddt_seconds` (*waveline.spotwave.Setup attribute*), 15
`decibel_to_volts()` (*in module waveline.utils*), 27
`discover()` (*waveline.conditionwave.ConditionWave class method*), 5
`discover()` (*waveline.spotwave.SpotWave class method*), 18
`duration` (*waveline.datatypes.AERRecord attribute*), 24

E

`enabled` (*waveline.conditionwave.Setup attribute*), 11
`energy` (*waveline.datatypes.AERRecord attribute*), 24

F

`filter_highpass_hz` (*waveline.conditionwave.Setup attribute*), 11
`filter_highpass_hz` (*waveline.spotwave.Setup attribute*), 15
`filter_lowpass_hz` (*waveline.conditionwave.Setup attribute*), 11

[filter_lowpass_hz](#) ([waveline.spotwave.Setup](#) attribute), 15
[filter_order](#) ([waveline.conditionwave.Setup](#) attribute), 11
[filter_order](#) ([waveline.spotwave.Setup](#) attribute), 15
[firmware_version](#) ([waveline.conditionwave.Info](#) attribute), 9
[firmware_version](#) ([waveline.spotwave.Info](#) attribute), 13
[flags](#) ([waveline.datatypes.AERRecord](#) attribute), 24
[fpga_version](#) ([waveline.conditionwave.Info](#) attribute), 10

G

[get_ae_data\(\)](#) ([waveline.conditionwave.ConditionWave](#) method), 8
[get_ae_data\(\)](#) ([waveline.spotwave.SpotWave](#) method), 20
[get_data\(\)](#) ([waveline.spotwave.SpotWave](#) method), 21
[get_data_log\(\)](#) ([waveline.spotwave.SpotWave](#) method), 21
[get_info\(\)](#) ([waveline.conditionwave.ConditionWave](#) method), 5
[get_info\(\)](#) ([waveline.spotwave.SpotWave](#) method), 18
[get_setup\(\)](#) ([waveline.conditionwave.ConditionWave](#) method), 5
[get_setup\(\)](#) ([waveline.spotwave.SpotWave](#) method), 18
[get_status\(\)](#) ([waveline.conditionwave.ConditionWave](#) method), 5
[get_status\(\)](#) ([waveline.spotwave.SpotWave](#) method), 18
[get_tr_data\(\)](#) ([waveline.conditionwave.ConditionWave](#) method), 8
[get_tr_data\(\)](#) ([waveline.spotwave.SpotWave](#) method), 20

I

[Info](#) (class in [waveline.conditionwave](#)), 9
[Info](#) (class in [waveline.spotwave](#)), 13
[input_range_decibel](#) ([waveline.spotwave.Info](#) attribute), 14

L

[log_data_usage](#) ([waveline.spotwave.Status](#) attribute), 22
[logging](#) ([waveline.spotwave.Setup](#) attribute), 14
[logging](#) ([waveline.spotwave.Status](#) attribute), 22

M

[max_sample_rate](#) ([waveline.conditionwave.Info](#) attribute), 10

[MAX_SAMPLERATE](#) ([waveline.conditionwave.ConditionWave](#) attribute), 5
[model](#) ([waveline.spotwave.Info](#) attribute), 14
[module](#)
 [waveline.conditionwave](#), 3
 [waveline.datatypes](#), 23
 [waveline.spotwave](#), 13
 [waveline.utils](#), 27

P

[PORT](#) ([waveline.conditionwave.ConditionWave](#) attribute), 5
[PRODUCT_ID](#) ([waveline.spotwave.SpotWave](#) attribute), 17

R

[range_count](#) ([waveline.conditionwave.Info](#) attribute), 10
[raw](#) ([waveline.datatypes.TRRecord](#) attribute), 25
[recording](#) ([waveline.spotwave.Setup](#) attribute), 14
[recording](#) ([waveline.spotwave.Status](#) attribute), 22
[rise_time](#) ([waveline.datatypes.AERRecord](#) attribute), 24

S

[samples](#) ([waveline.datatypes.TRRecord](#) attribute), 25
[set_cct\(\)](#) ([waveline.spotwave.SpotWave](#) method), 19
[set_channel\(\)](#) ([waveline.conditionwave.ConditionWave](#) method), 6
[set_continuous_mode\(\)](#) ([waveline.conditionwave.ConditionWave](#) method), 6
[set_continuous_mode\(\)](#) ([waveline.spotwave.SpotWave](#) method), 18
[set_datetime\(\)](#) ([waveline.spotwave.SpotWave](#) method), 20
[set_ddt\(\)](#) ([waveline.conditionwave.ConditionWave](#) method), 6
[set_ddt\(\)](#) ([waveline.spotwave.SpotWave](#) method), 19
[set_filter\(\)](#) ([waveline.conditionwave.ConditionWave](#) method), 7
[set_filter\(\)](#) ([waveline.spotwave.SpotWave](#) method), 20
[set_logging_mode\(\)](#) ([waveline.spotwave.SpotWave](#) method), 20
[set_range\(\)](#) ([waveline.conditionwave.ConditionWave](#) method), 6
[set_status_interval\(\)](#) ([waveline.conditionwave.ConditionWave](#) method), 6
[set_status_interval\(\)](#) ([waveline.spotwave.SpotWave](#) method), 19

`set_threshold()` (waveline.conditionwave.ConditionWave method), 7
`set_threshold()` (waveline.spotwave.SpotWave method), 20
`set_tr_decimation()` (waveline.conditionwave.ConditionWave method), 6
`set_tr_decimation()` (waveline.spotwave.SpotWave method), 19
`set_tr_enabled()` (waveline.conditionwave.ConditionWave method), 6
`set_tr_enabled()` (waveline.spotwave.SpotWave method), 19
`set_tr_postduration()` (waveline.conditionwave.ConditionWave method), 7
`set_tr_postduration()` (waveline.spotwave.SpotWave method), 19
`set_tr_pretrigger()` (waveline.conditionwave.ConditionWave method), 6
`set_tr_pretrigger()` (waveline.spotwave.SpotWave method), 19
`Setup` (class in waveline.conditionwave), 10
`Setup` (class in waveline.spotwave), 14
`SpotWave` (class in waveline.spotwave), 15
`start_acquisition()` (waveline.conditionwave.ConditionWave method), 7
`start_acquisition()` (waveline.spotwave.SpotWave method), 20
`start_pulsing()` (waveline.conditionwave.ConditionWave method), 7
`Status` (class in waveline.conditionwave), 12
`Status` (class in waveline.spotwave), 22
`status_interval_seconds` (waveline.conditionwave.Setup attribute), 11
`status_interval_seconds` (waveline.spotwave.Setup attribute), 15
`stop_acquisition()` (waveline.conditionwave.ConditionWave method), 7
`stop_acquisition()` (waveline.spotwave.SpotWave method), 20
`stop_pulsing()` (waveline.conditionwave.ConditionWave method), 7
`stream()` (waveline.conditionwave.ConditionWave method), 8
`stream()` (waveline.spotwave.SpotWave method), 21

T

`temperature` (waveline.conditionwave.Status attribute), 12
`temperature` (waveline.spotwave.Status attribute), 22
`threshold_volts` (waveline.conditionwave.Setup attribute), 11
`threshold_volts` (waveline.spotwave.Setup attribute), 15
`time` (waveline.datatypes.AERecord attribute), 24
`time` (waveline.datatypes.TRRecord attribute), 25
`tr_decimation` (waveline.conditionwave.Setup attribute), 11
`tr_decimation` (waveline.spotwave.Setup attribute), 15
`tr_enabled` (waveline.conditionwave.Setup attribute), 11
`tr_enabled` (waveline.spotwave.Setup attribute), 15
`tr_postduration_samples` (waveline.conditionwave.Setup attribute), 11
`tr_postduration_samples` (waveline.spotwave.Setup attribute), 15
`tr_pretrigger_samples` (waveline.conditionwave.Setup attribute), 11
`tr_pretrigger_samples` (waveline.spotwave.Setup attribute), 15
`tra_i` (waveline.datatypes.AERecord attribute), 24
`tra_i` (waveline.datatypes.TRRecord attribute), 25
`TRRecord` (class in waveline.datatypes), 24
`type_` (waveline.datatypes.AERecord attribute), 24
`type_` (waveline.spotwave.Info attribute), 13

V

`VENDOR_ID` (waveline.spotwave.SpotWave attribute), 17
`volts_to_decibel()` (in module waveline.utils), 27

W

`waveline.conditionwave` module, 3
`waveline.datatypes` module, 23
`waveline.spotwave` module, 13
`waveline.utils` module, 27