
waveline

Release 0.8.0

Lukas Berbuer (Vallen Systeme GmbH)

Mar 20, 2024

LIBRARY DOCUMENTATION

1	waveline.spotwave	3
1.1	SpotWave	3
2	waveline.linwave	11
2.1	LinWave	11
3	waveline.datatypes	19
3.1	AERecord	19
3.2	Info	20
3.3	Setup	21
3.4	Status	23
3.5	TRRecord	24
4	waveline.utils	25
4.1	decibel_to_volts	25
4.2	volts_to_decibel	25
5	Changelog	27
5.1	Unreleased	27
5.2	0.8.0 - 2024-03-20	27
5.3	0.7.1 - 2023-10-18	27
5.4	0.7.0 - 2023-10-17	28
5.5	0.6.0 - 2022-08-01	28
5.6	0.5.0 - 2022-06-21	29
5.7	0.4.1 - 2022-06-20	29
5.8	0.4.0 - 2022-05-17	29
5.9	0.3.0 - 2021-06-15	30
5.10	0.2.0 - 2020-12-18	30
6	Indices and tables	31
	Python Module Index	33
	Index	35

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

<i>waveline.spotwave</i>	Module for spotWave device.
<i>waveline.linwave</i>	Module for linWave device.
<i>waveline.datatypes</i>	Common datatypes.
<i>waveline.utils</i>	Utility functions.

WAVELINE.SPOTWAVE

Module for spotWave device.

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

Classes

<i>SpotWave</i> (port)	Interface for spotWave devices.
------------------------	---------------------------------

1.1 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, poll_interval_seconds])</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.
<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>get_tr_snapshot(samples[, raw])</code>	Read snapshot of transient data.
<code>identify()</code>	Blink LED to identify device.
<code>set_cct(interval_seconds)</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>start_pulsing([interval, count])</code>	Start pulsing.
<code>stop_acquisition()</code>	Stop acquisition.
<code>stop_pulsing()</code>	Stop pulsing.
<code>stream(*args, **kwargs)</code>	Alias for <i>SpotWave.acquire</i> method.

Attributes

<i>CLOCK</i>	Internal clock in Hz
<i>PRODUCT_ID</i>	USB product id of SpotWave device
<i>VENDOR_ID</i>	USB vendor id of Vallen Systeme GmbH
<i>connected</i>	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.

classmethod discover()

Discover connected spotWave devices.

Return type

List[str]

Returns

List of port names

identify()

Blink LED to identify device.

Note: Available since firmware version 00.2D.

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*)

Set coupling check transmitter (CCT) / pulser interval.

The pulser amplitude is 3.3 V.

Parameters**interval_seconds** (*float*) – Pulser interval in seconds. If < 0, the pulse is synchronized with the first sample of the *get_tr_snapshot* 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.

start_pulsing(*interval=1, count=0*)

Start pulsing.

The pulser amplitude is 3.3 V. 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

- **interval** (*float*) – Interval between pulses in seconds
- **count** (*int*) – Number of pulses per channel (should be even), 0 for infinite pulses

stop_pulsing()

Stop pulsing.

get_ae_data()

Get AE data records.

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

get_tr_snapshot(*samples, raw=False*)

Read snapshot of transient data.

The recording starts with the execution of the command. The *trai* and *time* of the returned records are always 0.

Parameters

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

Return type*TRRecord***Returns**

Transient data record

get_data(*samples, raw=False*)

Read snapshot of transient data.

The recording starts with the execution of the command.

Deprecated: Please use the *get_tr_snapshot* method instead.

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*)**acquire**(*raw=False*, *poll_interval_seconds=0.01*)

High-level method to continuously acquire data.

Parameters

- **raw** (*bool*) – Return TR amplitudes as ADC values if *True*, skip conversion to volts
- **poll_interval_seconds** (*float*) – Pause between data polls in seconds

Yields

AE and TR data records

Return type*Iterator*[*Union*[*AERecord*, *TRRecord*]]**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):
>>>             ...
```

stream(**args*, ***kwargs*)Alias for *SpotWave.acquire* method.Deprecated: Please use the *acquire* method instead.**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.

WAVELINE.LINWAVE

Module for linWave device.

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

Classes

<i>LinWave</i> (address)	Interface for linWave device.
--------------------------	-------------------------------

2.1 LinWave

class waveline.linwave.**LinWave**(address)

Interface for linWave 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 linWave devices.

Returns

Instance of *LinWave*

Example

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

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

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

2. Using the async context manager:

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


Methods

<code>__init__(address)</code>	Initialize device.
<code>acquire([raw, poll_interval_seconds])</code>	High-level method to continuously acquire data.
<code>close()</code>	Close connection.
<code>connect()</code>	Connect to device.
<code>discover([timeout])</code>	Discover linWave 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>get_tr_snapshot(channel, samples[, ...])</code>	Get snapshot of transient data.
<code>identify([channel])</code>	Blink LEDs to identify device or single channel.
<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_range_index(channel, range_index)</code>	Set input range by index.
<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>	Stop pulsing.
<code>stream(channel, blocksize, *[, raw, timeout])</code>	Async generator to stream channel data.

Attributes

<code>CHANNELS</code>	Available channels
<code>MAX_SAMPLERATE</code>	Maximum sampling rate in Hz
<code>PORT</code>	Control port number
<code>RANGES</code>	
<code>connected</code>	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 linWave 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.

async connect()

Connect to device.

async close()

Close connection.

async identify(*channel=0*)

Blink LEDs to identify device or single channel.

Parameters

channel (*int*) – Channel number (0 for all to identify device)

Note: Available since firmware version 2.10.

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.

Parameters

channel (*int*) – Channel number

Return type

Setup

async set_range_index(*channel, range_index*)

Set input range by index.

Retrieve selectable ranges with the *get_info* method.

Parameters

- **channel** (*int*) – Channel number (0 for all channels)
- **range_index** (*int*) – Input range index (0: 0.05 V, 1: 5 V)

async set_range(*channel*, *range_volts*)

Set input range.

Retrieve selectable ranges with the [get_info](#) method.

Parameters

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

Deprecated: Please use the [set_range_index](#) method instead.

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()

Stop pulsing.

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 get_tr_snapshot(channel, samples, pretrigger_samples=0, *, raw=False)

Get snapshot of transient data.

The recording starts with the execution of the command. The total number of samples is the sum of *samples* and the *pretrigger_samples*. The *trai* and *time* of the returned records are always 0.

Parameters

- **channel** (`int`) – Channel number (0 for all channels)
- **samples** (`int`) – Number of samples to read
- **pretrigger_samples** (`int`) – Number of samples to read before the execution of the command
- **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, poll_interval_seconds=0.05)

High-level method to continuously acquire data.

Parameters

- **raw** (`bool`) – Return TR amplitudes as ADC values if *True*, skip conversion to volts
- **poll_interval_seconds** (`float`) – Pause between data polls in seconds

Yields

AE and TR data records

Return type

`AsyncIterator[Union[AERecord, TRRecord]]`

Example

```

>>> async with waveline.LinWave("192.254.100.100") as lw:
>>>     # apply settings
>>>     await lw.set_channel(channel=1, enabled=True)
>>>     await lw.set_channel(channel=2, enabled=False)
>>>     await lw.set_range_index(channel=1, range_index=0) # 0: 50 mV
>>>     async for record in lw.acquire():
>>>         # do something with the data depending on the type
>>>         if isinstance(record, waveline.AERRecord):
>>>             ...
>>>         if isinstance(record, waveline.TRRecord):
>>>             ...

```

stream(channel, blocksize, *, raw=False, timeout=5)

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
- **timeout** (Optional[float]) – Timeout in seconds

Yields

Tuple of

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

Raises

TimeoutError – If TCP socket read exceeds *timeout*, usually because of buffer overflows

Return type

`AsyncIterator[Tuple[float, ndarray]]`

Example

```

>>> async with waveline.LinWave("192.168.0.100") as lw:
>>>     # apply settings
>>>     await lw.set_range_index(channel=1, range_index=0) # 0: 50 mV
>>>     await lw.set_filter(channel=1, highpass=100e3, lowpass=500e3, order=8)
>>>     # open streaming port before start acq afterwards (order matters!)
>>>     stream = lw.stream(channel=1, blocksize=65536)
>>>     await lw.start_acquisition()
>>>     async for time, block in stream:
>>>         # do something with the data
>>>         ...

```

WAVELINE.DATATYPES

Common datatypes.

Classes

<i>AERecord</i> (type_, channel, time, amplitude, ...)	AE data record, either status or hit data.
<i>Info</i> (hardware_id, firmware_version, ...)	Device information (static).
<i>Setup</i> (enabled, input_range, adc_to_volts, ...)	Channel setup.
<i>Status</i> (temperature, recording, pulsing, extra)	Status information.
<i>TRRecord</i> (channel, trai, time, samples, data)	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.

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

Methods

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

Attributes

<code>type_</code>	Record type (<i>H</i> for hit or <i>S</i> for status data)
<code>channel</code>	Channel number
<code>time</code>	Time in seconds (since <i>start_acq</i> command)
<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 <i>AERecord</i> and <i>TRRecord</i>)
<code>flags</code>	Hit flags

type_: `str`

Record type (*H* for hit or *S* for status data)

channel: `int`

Channel number

time: `float`

Time in seconds (since *start_acq* command)

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 Info

```
class waveline.datatypes.Info(hardware_id, firmware_version, channel_count, input_range, adc_to_volts,  
                             extra)
```

Device information (static).

```
__init__(hardware_id, firmware_version, channel_count, input_range, adc_to_volts, extra)
```


Methods

```
__init__(hardware_id, firmware_version, ...)
```

Attributes

<code>hardware_id</code>	Unique hardware id
<code>firmware_version</code>	Firmware version
<code>channel_count</code>	Number of channels
<code>input_range</code>	List of selectable input ranges in human-readable format
<code>adc_to_volts</code>	Conversion factors from ADC values to V for all input ranges
<code>extra</code>	Extra device information (specific to device and firmware version)

hardware_id: `Optional[str]`

Unique hardware id

firmware_version: `str`

Firmware version

channel_count: `int`

Number of channels

input_range: `List[str]`

List of selectable input ranges in human-readable format

adc_to_volts: `List[float]`

Conversion factors from ADC values to V for all input ranges

extra: `Dict[str, str]`

Extra device information (specific to device and firmware version)

3.3 Setup

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

Channel setup.

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

Methods

```
__init__(enabled, input_range, adc_to_volts, ...)
```

Attributes

<code>enabled</code>	Flag if channel is enabled
<code>input_range</code>	Input range index of <i>Info.input_range</i> list
<code>adc_to_volts</code>	Conversion factor from ADC values to volts
<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>continuous_mode</code>	Flag if continuous mode is enabled
<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>tr_enabled</code>	Flag if 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>extra</code>	Extra setup information (specific to device and firmware version)

enabled: `bool`

Flag if channel is enabled

input_range: `int`

Input range index of *Info.input_range* list

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

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 if 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

extra: `Dict[str, str]`
 Extra setup information (specific to device and firmware version)

3.4 Status

class waveline.datatypes.**Status**(*temperature, recording, pulsing, extra*)

Status information.

__init__(*temperature, recording, pulsing, extra*)

Methods

__init__(*temperature, recording, pulsing, extra*)

Attributes

<i>temperature</i>	Device temperature in °C
<i>recording</i>	Flag if acquisition is active
<i>pulsing</i>	Flag if pulsing is active
<i>extra</i>	Extra status information (specific to device and firmware version)

temperature: `float`
 Device temperature in °C

recording: `bool`
 Flag if acquisition is active

pulsing: `bool`
 Flag if pulsing is active

extra: `Dict[str, str]`
 Extra status information (specific to device and firmware version)

3.5 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

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

channel: *int*

Channel number

trai: *int*

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

time: *float*

Time in seconds (since *start_acq* command)

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

Utility functions.

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 0.8.0 - 2024-03-20

5.2.1 Added

- [spotwave] `SpotWave.get_tr_snapshot` method
- [spotwave] `SpotWave.start_pulsing`, `SpotWave.stop_pulsing` methods

5.2.2 Changed

- Use common `Info`, `Status` and `Setup` dataclass for both `spotWave` and `linWave`
- [linwave] Deprecate `LinWave.set_range` method, use `LinWave.set_range_index` instead
- [linwave] Remove `Info.max_sample_rate` field
- [spotwave] Deprecate `SpotWave.get_data`, use `SpotWave.get_tr_snapshot` instead
- [spotwave] Remove `sync` parameter of `SpotWave.set_cct` method
- Remove deprecated `conditionwave` module

5.3 0.7.1 - 2023-10-18

5.3.1 Fixed

- [spotwave] Return device names/paths from `SpotWave.discover` method, e.g. `/dev/ttyACM0` instead of `ttyACM0` on Linux systems

5.4 0.7.0 - 2023-10-17

5.4.1 Added

- [linwave] `LinWave.set_range_index` method
- [linwave] `LinWave.identify` method to blind all LEDs or single channel to identify device/channel
- [linwave] `LinWave.get_tr_snapshot` method (experimental)
- [linwave] Add `hardware_id` to `get_info` output
- [spotwave] `LinWave.identify` method to blind LED
- Add Python 3.11 and 3.12 to CI pipeline
- Examples:
 - `linwave_pulsing`
 - `linwave_cont_tr`

5.4.2 Changed

- [spotwave] Remove `cct_seconds` field in `get_setup` response Setup
- [spotwave] Make `readlines` method private
- List input range(s) in `get_info` response `Info.input_range` (human-readable format)
 - Remove `linwave.Info.range_count` field
 - Remove `spotwave.Info.input_range_decibel` field

5.5 0.6.0 - 2022-08-01

5.5.1 Changed

- [linwave] Timeout parameter for `LinWave.stream` method (to detect buffer overflows), default: 5 s

5.5.2 Fixed

- [linwave] Discovery port binding of client
- [linwave] Reduce CPU load in stream by setting TCP limit/buffer

5.6 0.5.0 - 2022-06-21

5.6.1 Added

- `poll_interval_seconds` parameter for `SpotWave.acquire` and `ConditionWave.acquire` / `LinWave.acquire` method

5.6.2 Changed

- Rename `ConditionWave` to `LinWave` (and the corresponding module `conditionwave` to `linwave`). The `ConditionWave` class is still an alias for the `LinWave` class but deprecated and will be removed in the future

5.6.3 Fixed

- [linwave] Fix timeouts for multiline responses (`get_info`, `get_status`, `get_setup`)

5.7 0.4.1 - 2022-06-20

5.7.1 Fixed

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

5.8 0.4.0 - 2022-05-17

5.8.1 Added

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

5.8.2 Changed

- [linwave] Rename `set_decimation` method to `set_tr_decimation`
- [linwave] 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

5.8.3 Fixed

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

5.9 0.3.0 - 2021-06-15

5.9.1 Added

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

5.9.2 Changed

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

5.9.3 Removed

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

5.9.4 Fixed

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

5.10 0.2.0 - 2020-12-18

First public release

INDICES AND TABLES

- `genindex`
- `modindex`

PYTHON MODULE INDEX

W

`waveline.datatypes`, [19](#)

`waveline.linwave`, [11](#)

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

`waveline.utils`, [25](#)

Symbols

__init__() (waveline.datatypes.AERRecord method), 19
 __init__() (waveline.datatypes.Info method), 20
 __init__() (waveline.datatypes.Setup method), 21
 __init__() (waveline.datatypes.Status method), 23
 __init__() (waveline.datatypes.TRRecord method), 24
 __init__() (waveline.linwave.LinWave method), 11
 __init__() (waveline.spotwave.SpotWave method), 3

A

acquire() (waveline.linwave.LinWave method), 17
 acquire() (waveline.spotwave.SpotWave method), 9
 adc_to_volts (waveline.datatypes.Info attribute), 21
 adc_to_volts (waveline.datatypes.Setup attribute), 22
 AERRecord (class in waveline.datatypes), 19
 amplitude (waveline.datatypes.AERRecord attribute), 20

C

channel (waveline.datatypes.AERRecord attribute), 20
 channel (waveline.datatypes.TRRecord attribute), 24
 channel_count (waveline.datatypes.Info attribute), 21
 CHANNELS (waveline.linwave.LinWave attribute), 13
 clear_buffer() (waveline.spotwave.SpotWave method), 5
 clear_data_log() (waveline.spotwave.SpotWave method), 9
 CLOCK (waveline.spotwave.SpotWave attribute), 5
 close() (waveline.linwave.LinWave method), 14
 close() (waveline.spotwave.SpotWave method), 5
 connect() (waveline.linwave.LinWave method), 14
 connect() (waveline.spotwave.SpotWave method), 5
 connected (waveline.linwave.LinWave property), 14
 connected (waveline.spotwave.SpotWave property), 5
 continuous_mode (waveline.datatypes.Setup attribute), 22
 counts (waveline.datatypes.AERRecord attribute), 20

D

data (waveline.datatypes.TRRecord attribute), 24
 ddt_seconds (waveline.datatypes.Setup attribute), 22
 decibel_to_volts() (in module waveline.utils), 25

discover() (waveline.linwave.LinWave class method), 13
 discover() (waveline.spotwave.SpotWave class method), 5
 duration (waveline.datatypes.AERRecord attribute), 20

E

enabled (waveline.datatypes.Setup attribute), 22
 energy (waveline.datatypes.AERRecord attribute), 20
 extra (waveline.datatypes.Info attribute), 21
 extra (waveline.datatypes.Setup attribute), 23
 extra (waveline.datatypes.Status attribute), 23

F

filter_highpass_hz (waveline.datatypes.Setup attribute), 22
 filter_lowpass_hz (waveline.datatypes.Setup attribute), 22
 filter_order (waveline.datatypes.Setup attribute), 22
 firmware_version (waveline.datatypes.Info attribute), 21
 flags (waveline.datatypes.AERRecord attribute), 20

G

get_ae_data() (waveline.linwave.LinWave method), 17
 get_ae_data() (waveline.spotwave.SpotWave method), 8
 get_data() (waveline.spotwave.SpotWave method), 8
 get_data_log() (waveline.spotwave.SpotWave method), 9
 get_info() (waveline.linwave.LinWave method), 14
 get_info() (waveline.spotwave.SpotWave method), 5
 get_setup() (waveline.linwave.LinWave method), 14
 get_setup() (waveline.spotwave.SpotWave method), 5
 get_status() (waveline.linwave.LinWave method), 14
 get_status() (waveline.spotwave.SpotWave method), 6
 get_tr_data() (waveline.linwave.LinWave method), 17
 get_tr_data() (waveline.spotwave.SpotWave method), 8
 get_tr_snapshot() (waveline.linwave.LinWave method), 17

`get_tr_snapshot()` (*waveline.spotwave.SpotWave method*), 8

H

`hardware_id` (*waveline.datatypes.Info attribute*), 21

I

`identify()` (*waveline.linwave.LinWave method*), 14

`identify()` (*waveline.spotwave.SpotWave method*), 5

`Info` (*class in waveline.datatypes*), 20

`input_range` (*waveline.datatypes.Info attribute*), 21

`input_range` (*waveline.datatypes.Setup attribute*), 22

L

`LinWave` (*class in waveline.linwave*), 11

M

`MAX_SAMPLERATE` (*waveline.linwave.LinWave attribute*), 13

`module`

`waveline.datatypes`, 19

`waveline.linwave`, 11

`waveline.spotwave`, 3

`waveline.utils`, 25

P

`PORT` (*waveline.linwave.LinWave attribute*), 13

`PRODUCT_ID` (*waveline.spotwave.SpotWave attribute*), 5

`pulsing` (*waveline.datatypes.Status attribute*), 23

R

`raw` (*waveline.datatypes.TRRecord attribute*), 24

`recording` (*waveline.datatypes.Status attribute*), 23

`rise_time` (*waveline.datatypes.AERRecord attribute*), 20

S

`samples` (*waveline.datatypes.TRRecord attribute*), 24

`set_cct()` (*waveline.spotwave.SpotWave method*), 7

`set_channel()` (*waveline.linwave.LinWave method*), 15

`set_continuous_mode()` (*waveline.linwave.LinWave method*), 15

`set_continuous_mode()` (*waveline.spotwave.SpotWave method*), 6

`set_datetime()` (*waveline.spotwave.SpotWave method*), 7

`set_ddt()` (*waveline.linwave.LinWave method*), 15

`set_ddt()` (*waveline.spotwave.SpotWave method*), 6

`set_filter()` (*waveline.linwave.LinWave method*), 16

`set_filter()` (*waveline.spotwave.SpotWave method*), 7

`set_logging_mode()` (*waveline.spotwave.SpotWave method*), 7

`set_range()` (*waveline.linwave.LinWave method*), 14

`set_range_index()` (*waveline.linwave.LinWave method*), 14

`set_status_interval()` (*waveline.linwave.LinWave method*), 15

`set_status_interval()` (*waveline.spotwave.SpotWave method*), 6

`set_threshold()` (*waveline.linwave.LinWave method*), 16

`set_threshold()` (*waveline.spotwave.SpotWave method*), 7

`set_tr_decimation()` (*waveline.linwave.LinWave method*), 15

`set_tr_decimation()` (*waveline.spotwave.SpotWave method*), 6

`set_tr_enabled()` (*waveline.linwave.LinWave method*), 15

`set_tr_enabled()` (*waveline.spotwave.SpotWave method*), 6

`set_tr_postduration()` (*waveline.linwave.LinWave method*), 16

`set_tr_postduration()` (*waveline.spotwave.SpotWave method*), 7

`set_tr_pretrigger()` (*waveline.linwave.LinWave method*), 16

`set_tr_pretrigger()` (*waveline.spotwave.SpotWave method*), 7

`Setup` (*class in waveline.datatypes*), 21

`SpotWave` (*class in waveline.spotwave*), 3

`start_acquisition()` (*waveline.linwave.LinWave method*), 16

`start_acquisition()` (*waveline.spotwave.SpotWave method*), 7

`start_pulsing()` (*waveline.linwave.LinWave method*), 16

`start_pulsing()` (*waveline.spotwave.SpotWave method*), 7

`Status` (*class in waveline.datatypes*), 23

`status_interval_seconds` (*waveline.datatypes.Setup attribute*), 22

`stop_acquisition()` (*waveline.linwave.LinWave method*), 16

`stop_acquisition()` (*waveline.spotwave.SpotWave method*), 7

`stop_pulsing()` (*waveline.linwave.LinWave method*), 17

`stop_pulsing()` (*waveline.spotwave.SpotWave method*), 8

`stream()` (*waveline.linwave.LinWave method*), 18

`stream()` (*waveline.spotwave.SpotWave method*), 9

T

`temperature` (*waveline.datatypes.Status attribute*), 23

`threshold_volts` (*waveline.datatypes.Setup attribute*), 22

[time \(waveline.datatypes.AERecord attribute\), 20](#)
[time \(waveline.datatypes.TRRecord attribute\), 24](#)
[tr_decimation \(waveline.datatypes.Setup attribute\), 23](#)
[tr_enabled \(waveline.datatypes.Setup attribute\), 23](#)
[tr_postduration_samples \(waveline.datatypes.Setup attribute\), 23](#)
[tr_pretrigger_samples \(waveline.datatypes.Setup attribute\), 23](#)
[tra_i \(waveline.datatypes.AERecord attribute\), 20](#)
[tra_i \(waveline.datatypes.TRRecord attribute\), 24](#)
[TRRecord \(class in waveline.datatypes\), 24](#)
[type_ \(waveline.datatypes.AERecord attribute\), 20](#)

V

[VENDOR_ID \(waveline.spotwave.SpotWave attribute\), 5](#)
[volts_to_decibel\(\) \(in module waveline.utils\), 25](#)

W

[waveline.datatypes module, 19](#)
[waveline.linwave module, 11](#)
[waveline.spotwave module, 3](#)
[waveline.utils module, 25](#)