waveline

Release 0.7.1

Lukas Berbuer (Vallen Systeme GmbH)

Oct 18, 2023

LIBRARY DOCUMENTATION

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

waveline.spotwave	Module for spotWave device.
waveline.linwave	Module for linWave device.
waveline.conditionwave	Module for conditionWave device (alias for *waveline.linwave*, *deprecated*).
waveline.datatypes	Common datatypes.
waveline.utils	Utility functions.

CHAPTER

ONE

WAVELINE.SPOTWAVE

Module for spotWave device.

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

Classes

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

1.1 Info

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

Device information.

__init__(*firmware_version*, *type_*, *model*, *input_range*)

Methods

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

Attributes

firmware_version	Firmware version (major, minor)
*type_*	Device type
model	Model identifier
input_range	Input range

firmware_version: str

Firmware version (major, minor)

type_: str
Device type

model: str
Model identifier

input_range: str
Input range

1.2 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*)

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*)

Methods

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

Attributes

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

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

1.3 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

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

Attributes

CLOCK	Internal clock in Hz
PRODUCT_ID	USB product id of SpotWave device
VENDOR_ID	USB vendor id of Vallen Systeme GmbH
connected	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 continous 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 ransmitter (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.

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*, *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 us 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.

1.4 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

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

CHAPTER

TWO

WAVELINE.LINWAVE

Module for linWave device.

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

Classes

Info(hardware_id, firmware_version, ...)	Device information.
LinWave(address)	Interface for linWave device.
Setup(adc_range_volts, adc_to_volts, ...)	Setup.
Status(temperature, buffer_size)	Status information.

2.1 Info

class waveline.linwave.**Info**(*hardware_id*, *firmware_version*, *fpga_version*, *channel_count*, *input_range*, *max_sample_rate*, *adc_to_volts*)

Device information.

__init__(*hardware_id*, *firmware_version*, *fpga_version*, *channel_count*, *input_range*, *max_sample_rate*, *adc_to_volts*)

Methods

__init__(hardware_id, firmware_version, ...)	

Attributes

hardware_id	Unique hardware id (since firmware version 2.13)
firmware_version	Firmware version
fpga_version	FPGA version
channel_count	Number of channels
input_range	List of selectable input ranges
max_sample_rate	Max sampling rate
adc_to_volts	Conversion factors from ADC values to V for both ranges

hardware_id: `str`
Unique hardware id (since firmware version 2.13)

firmware_version: `str`
Firmware version

fpga_version: `str`
FPGA version

channel_count: `int`
Number of channels

input_range: `List[str]`
List of selectable input ranges

max_sample_rate: `float`
Max sampling rate

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

2.2 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

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

Attributes

CHANNELS	Available channels
MAX_SAMPLERATE	Maximum sampling rate in Hz
PORT	Control port number
RANGES	
connected	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 adresses

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 (0 for all channels)

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_range_index(*channel*, *range_index*)

Set input range by index.

Parameters

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

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 continous 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 µs

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 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(channel=1, range_volts=0.05)
>>>     async for record in lw.acquire():
>>>         # do something with the data depending on the type
>>>         if isinstance(record, waveline.AERecord):
>>>             ...
>>>         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(0.05)
>>>     await lw.set_filter(100e3, 500e3, 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
>>>         ...
```

2.3 Setup

class waveline.linwave.**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

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

2.4 Status

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

Status information.

__init__(*temperature*, *buffer_size*)

Methods

__init__(temperature, buffer_size)	

Attributes

temperature	Device temperature in °C
buffer_size	Buffer size in bytes

temperature: float

Device temperature in °C

buffer_size: int

Buffer size in bytes

CHAPTER

THREE

WAVELINE.CONDITIONWAVE

Module for conditionWave device (alias for *waveline.linwave*, *deprecated*).

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

Classes

ConditionWave(*args, **kwargs)	Interface for conditionWave device.

3.1 ConditionWave

class waveline.conditionwave.**ConditionWave**(**args*, ***kwargs*)

Interface for conditionWave device.

Deprecated:
Please use `LinWave` class instead. The *ConditionWave* class will be removed in the future.

__init__(**args*, ***kwargs*)

Initialize device.

Parameters
address – 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

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

Attributes

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

CHAPTER

FOUR

WAVELINE.DATATYPES

Common datatypes.

Classes

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

4.1 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

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

trai: *int*
Transient recorder index (key between *AERecord* and *TRRecord*)

flags: *int*
Hit flags

4.2 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

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

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)

CHAPTER

FIVE

WAVELINE.UTILS

Utility functions.

Functions

decibel_to_volts(decibel)	Convert from dB(AE) to volts.
volts_to_decibel(volts)	Convert from volts to dB(AE).

5.1 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

5.2 volts_to_decibel

waveline.utils.**volts_to_decibel**(*volts*)

Convert from volts to dB(AE).

Parameters
volts (Union[float, ndarray]) – Inpult in volts, scalar or array

Return type
Union[float, ndarray]

Returns
Input value(s) in dB(AE)

CHAPTER

SIX

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.

6.1 Unreleased

6.2 0.7.1 - 2023-10-18

6.2.1 Fixed

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

6.3 0.7.0 - 2023-10-17

6.3.1 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`

6.3.2 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

6.4 0.6.0 - 2022-08-01

6.4.1 Changed

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

6.4.2 Fixed

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

6.5 0.5.0 - 2022-06-21

6.5.1 Added

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

6.5.2 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

6.5.3 Fixed

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

6.6 0.4.1 - 2022-06-20

6.6.1 Fixed

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

6.7 0.4.0 - 2022-05-17

6.7.1 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

6.7.2 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

6.7.3 Fixed

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

6.8 0.3.0 - 2021-06-15

6.8.1 Added

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

6.8.2 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

6.8.3 Removed

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

6.8.4 Fixed

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

6.9 0.2.0 - 2020-12-18

First public release

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex

PYTHON MODULE INDEX

W

INDEX

Symbols

A

B

C

D

E

F

G

H

I

L

M

P

R

S

T

V

W