waveline

Release 0.4.0

Lukas Berbuer (Vallen Systeme GmbH)

May 17, 2022

LIBRARY DOCUMENTATION

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

waveline.conditionwave	Module for conditionWave device.
waveline.spotwave	Module for spotWave device.

CHAPTER

ONE

WAVELINE.CONDITIONWAVE

Module for conditionWave device.

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

Classes

ConditionWave(address)	Interface for conditionWave device.
Info(firmware_version, fpga_version, ...)	Device information.
Setup(adc_range_volts, adc_to_volts, ...)	Setup.
Status(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

__init__(address)	Initialize device.
acquire([raw])	High-level method to continuously acquire data.
close()	Close connection.
connect()	Connect to device.
discover([timeout])	Discover conditionWave 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.
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_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])	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 conditionWave 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.

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 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 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)
>>>     # start daq and streaming
>>>     await cw.start_acquisition()
>>>     async for time, block in cw.stream(channel=1, blocksize=65536):
>>>         # 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

firmware_version	Firmware version
fpga_version	FPGA version
channel_count	Number of channels
range_count	Number of selectable ranges
max_sample_rate	Max sampling rate
adc_to_volts	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

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

1.4 Status

class waveline.conditionwave.**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

TWO

WAVELINE.SPOTWAVE

Module for spotWave device.

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

Classes

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

firmware_version	Firmware version (major, minor)
*type_*	Device type
model	Model identifier
input_range_decibel	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

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
cct_seconds	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

__init__(port)	Initialize device.
acquire([raw])	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.
readlines()	

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.

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 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.

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 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.

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

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.datetime

Device datetime

CHAPTER

THREE

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.

3.1 Unreleased

3.1.1 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

3.1.2 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

3.1.3 Fixed

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

3.2 0.3.0 - 2021-06-15

3.2.1 Added

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

3.2.2 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

3.2.3 Removed

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

3.2.4 Fixed

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

3.3 0.2.0 - 2020-12-18

First public release

CHAPTER

FOUR

TODOS

Todo:

- Implement parsing of record start marker

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pywaveline/envs/0.4.0/lib/python3.7/site-packages/waveline/spotwave.py:docstring of waveline.spotwave.SpotWave.get_ae_data, line 3.)

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex

PYTHON MODULE INDEX

W

INDEX

Symbols

A

B

C

D

E

F

G

I

L

M

P

R

S

T

V

W