
PyAbacus Documentation

Release 1.1.0

Tausand Electronica

Apr 23, 2020

CONTENTS

1	Contents	3
1.1	pyAbacus.core	3
1.2	pyAbacus.exceptions	7
1.3	pyAbacus.constants	7
2	Indices and tables	9
	Python Module Index	11
	Index	13



pyAbacus was build to simplify the usage of [Tausands](#) tools.

CONTENTS

1.1 pyAbacus.core

```
class pyAbacus.core.AbacusSerial (port)
    Builds a serial port from pyserial.

    findIdn ()

    flush ()

    getIdn ()

    getNChannels ()

    readSerial ()

    testDevice ()

    writeSerial (command, address, data_16o32)

class pyAbacus.core.CountersValues (n_channels)

    getCountersID ()

    getNumericAddresses ()

    getTimeLeft ()

    getValue (channel)

    getValues (channels)

    getValuesFormatted (channels)

    setCountersID (id)

    setTimeLeft (time)

    setValueFromArray (address, value)

    time_left = None
        in ms

class pyAbacus.core.Settings2Ch

    getAddressAndValue (timer)

    getSetting (timer)

    getSettingStr (timer)
```

setSetting (*setting, value*)

class `pyAbacus.core.Settings48Ch`

4 and 8 channel devices use as time base a second. Nevertheless 2 channel uses ns for all timers with the exception of the sampling time (ms).

exponentRepresentationToValue (*c, e*)

exponentsToBits (*c, e*)

fromBitsToValue (*bits*)

getAddressAndValue (*timer*)

getChannels ()

getSetting (*timer*)

For all timers: returns nanoseconds, for sampling returns ms.

getSettingStr (*timer*)

initAddresses ()

setSetting (*setting, value*)

For all timers: value is in nanoseconds, for sampling in ms.

valueToExponentRepresentation (*number*)

class `pyAbacus.core.Settings4Ch`

4 and 8 channel devices use as time base a second. Nevertheless 2 channel uses ns for all timers with the exception of the sampling time (ms).

class `pyAbacus.core.Settings8Ch`

4 and 8 channel devices use as time base a second. Nevertheless 2 channel uses ns for all timers with the exception of the sampling time (ms).

class `pyAbacus.core.Stream` (*abacus_port, counters, output_function=<built-in function print>*)

setCounters (*counters*)

start ()

stop ()

`pyAbacus.core.close` (*abacus_port*)

Closes a Tausand Abacus device session

`pyAbacus.core.dataArraysToCounters` (*abacus_port, addresses, data*)

Saves in local memory the values of device's counters.

Args: *abacus_port*: device port.

addresses: list of integers with device's register addresses.

data: list of integers with device's register values.

Returns: List of counter values as registered within the device.

`pyAbacus.core.dataArraysToSettings` (*abacus_port, addresses, data*)

Saves in local memory the values of device's settings.

Args: *abacus_port*: device port.

addresses: list of integers with device's register addresses.

data: list of integers with device's register values.

Returns: List of settings as registered within the device.

`pyAbacus.core.dataStreamToDataArrays (input_string, chunk_size=3)`

Builds data from string read on serial port.

Args: `input_string`: stream of bytes to convert. Should have the appropriate format, as given by a Tausand Abacus device.

`chunk_size` : integer, number of bytes per single data row. Use `chunk_size=3` for devices with inner 16-bit registers e.g. Tausand Abacus AB1002, where byte streams are: {address,MSB,LSB}. Use `chunk_size=5` for devices with inner 32-bit registers e.g. Tausand Abacus AB1004, where byte streams are: {address,MSB,2nd-MSB,2nd-LSB,LSB}.

Returns: Two lists of integer values: addresses, data.

Raises: `AbacusError`: Input string is not valid chunk size must either be 3 or 5.

`pyAbacus.core.findDevices (print_on=True)`

Returns a list of connected and available devices that match with a Tausand Abacus.

Scans all serial ports, and asks each of them their descriptions. When a device responds with a valid string, e.g. "Tausand Abacus AB1002", the port is included in the final answer.

Args: `print_on`: bool When True, prints devices information.

Returns: `ports`, `len(ports)` List of valid ports, and its length.

`pyAbacus.core.getAllCounters (abacus_port)`

Reads all counters from a Tausand Abacus device.

With a single call, this function reads all the counters within the device, including single-channel counters, 2-fold coincidence counters and multi-fold coincidence counters.

Example: `counters, counters_id = getAllCounters('COM3')`

Reads data from the device in port 'COM3', and might return for example,

`counters = [A:1023, B:1038, AB: 201]`

`counters_id = 37`

meaning that this is the 37th measurement made by the device, and the measurements were 1023 counts in A, 1038 counts in B, and 201 coincidences between A and B.

Args: `abacus_port`: device port.

Returns: List of counter values as registered within the device, and the sequential number of the reading.

`pyAbacus.core.getAllSettings (abacus_port)`

Reads all settings from a Tausand Abacus device.

With a single call, this function reads all the settings within the device, including sampling time, coincidence window, delay per channel and sleep time per channel.

Example: `settings = getAllCounters('COM3')`

Reads settings from the device in port 'COM3', and might return for example,

`[sampling:1000, delay_A:0, delay_B:0]`

meaning that sampling time is 1000ms, and no delay is added in channels A or B.

Args: `abacus_port`: device port.

Returns: List of settings as registered within the device.

`pyAbacus.core.getChannelFromName (name)`

Returns the number of input channels by reading the device name.

For example, if name="Tausand Abacus AB1004", returns 4.

Args: name: idn string of the device.

Returns: integer, number of input channels in device.

Raises: AbacusError: Not a valid abacus.

`pyAbacus.core.getCountersID (abacus_port)`

Reads the *counters_id* (consecutive number of measurements) in a Tausand Abacus.

When a new configuration is set, *counters_id*=0, indicating no valid data is available.

Each time a new set of valid measurements is available, *counters_id* increments 1 unit.

counters_id overflows at 1 million, starting over at *counters_id*=1.

Args: abacus_port: device port.

Returns: integer, *counters_id* value.

`pyAbacus.core.getFollowingCounters (abacus_port, counters)`

`pyAbacus.core.getIdn (abacus_port)`

Reads the identifier string model (IDN) from a Tausand Abacus.

Example: myidn = getIdn('COM3')

might return myidn = "Tausand Abacus AB1002"

Args: abacus_port: device port.

Returns: IDN string.

`pyAbacus.core.getSetting (abacus_port, setting)`

Get a single configuration setting within a Tausand Abacus.

Args: abacus_port: device port

setting: name of the setting to be written. Valid strings are: "sampling", "coincidence_window", "delay_N", "sleep_N", where "N" refers to a channel (A,B,C,D,...).

Returns: value for the setting. For "sampling", value in ms; for other settings, value in ns.

`pyAbacus.core.getTimeLeft (abacus_port)`

Reads the remaining time for the next measurement to be ready, in ms.

Args: abacus_port: device port

Returns: integer, in ms, of time left for next measurement.

`pyAbacus.core.open (abacus_port)`

Opens a session to a Tausand Abacus device

`pyAbacus.core.readSerial (abacus_port)`

Reads bytes available at the specified serial port.

`pyAbacus.core.renameDuplicates (old)`

`pyAbacus.core.setAllSettings (abacus_port, new_settings)`

`pyAbacus.core.setSetting (abacus_port, setting, value)`

Sets a configuration setting within a Tausand Abacus.

Args: abacus_port: device port

setting: name of the setting to be written. Valid strings are: “sampling”, “coincidence_window”, “delay_N”, “sleep_N”, where “N” refers to a channel (A,B,C,D,...).

value: new value for the setting. For “sampling”, value in ms; for other settings, value in ns.

`pyAbacus.core.writeSerial(abacus_port, command, address, data_16o32)`

Low level function. Writes in the specified serial port an instruction built based on command, memory address and data.

1.2 pyAbacus.exceptions

exception `pyAbacus.exceptions.AbacusError(message=“”)`

An unexpected error occurred.

exception `pyAbacus.exceptions.BaseError(message)`

exception `pyAbacus.exceptions.CheckSumError`

An error occurred while doing check sum.

exception `pyAbacus.exceptions.InvalidValueError(message=“”)`

The selected value is not valid

exception `pyAbacus.exceptions.TimeoutError(message=“”)`

A time out error occurred

1.3 pyAbacus.constants

`pyAbacus.constants.ADDRESS_DIRECTORY_2CH = {'coincidence_window_ms': 22, 'coincidence_windo`

Memory addresses

`pyAbacus.constants.BAUDRATE = 115200`

Default baudrate for the serial port communication

`pyAbacus.constants.BOUNCE_TIMEOUT = 1`

Number of times a specific transmission is tried

`pyAbacus.constants.COINCIDENCE_WINDOW_DEFAULT_VALUE = 10`

Default coincidence window time value (ns).

`pyAbacus.constants.COINCIDENCE_WINDOW_MAXIMUM_VALUE = 10000`

Maximum coincidence window time value (ns).

`pyAbacus.constants.COINCIDENCE_WINDOW_MINIMUM_VALUE = 5`

Minimum coincidence window time value (ns).

`pyAbacus.constants.COINCIDENCE_WINDOW_STEP_VALUE = 5`

Increase ratio on the coincidence window time value (ns).

`pyAbacus.constants.COUNTERS_VALUES = {}`

Global counters values variable

`pyAbacus.constants.CURRENT_OS = 'linux'`

Current operative system

`pyAbacus.constants.DELAY_DEFAULT_VALUE = 0`

Default delay time value (ns).

`pyAbacus.constants.DELAY_MAXIMUM_VALUE = 100`
Maximum delay time value (ns).

`pyAbacus.constants.DELAY_MINIMUM_VALUE = 0`
Minimum delay time value (ns).

`pyAbacus.constants.DELAY_STEP_VALUE = 5`
Increase ratio on the delay time value (ns).

`pyAbacus.constants.END_COMMUNICATION = 4`
End of message

`pyAbacus.constants.MAXIMUM_WRITING_TRIES = 20`
Number of tries done to write a value

`pyAbacus.constants.READ_VALUE = 14`
Reading operation signal

`pyAbacus.constants.SAMPLING_DEFAULT_VALUE = 1000`
Default sampling time value (ms)

`pyAbacus.constants.SAMPLING_VALUES = [1, 2, 5, 10, 20, 50, 100, 200, 500, 1000, 2000, 5000]`
From (1, 2, 5) ms to 1000 s

`pyAbacus.constants.SETTINGS = {}`
Global settings variable

`pyAbacus.constants.SLEEP_DEFAULT_VALUE = 0`
Default sleep time value (ns).

`pyAbacus.constants.SLEEP_MAXIMUM_VALUE = 100`
Maximum sleep time value (ns).

`pyAbacus.constants.SLEEP_MINIMUM_VALUE = 0`
Minimum sleep time value (ns).

`pyAbacus.constants.SLEEP_STEP_VALUE = 5`
Increase ratio on the sleep time value (ns).

`pyAbacus.constants.START_COMMUNICATION = 2`
Begin message signal

`pyAbacus.constants.TIMEOUT = 0.5`
Maximum time without answer from the serial port

`pyAbacus.constants.WRITE_VALUE = 15`
Writing operation signal

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

PYTHON MODULE INDEX

p

`pyAbacus.constants`, [7](#)
`pyAbacus.core`, [3](#)
`pyAbacus.exceptions`, [7](#)

A

AbacusError, 7
 AbacusSerial (class in *pyAbacus.core*), 3
 ADDRESS_DIRECTORY_2CH (in module *pyAbacus.constants*), 7

B

BaseError, 7
 BAUDRATE (in module *pyAbacus.constants*), 7
 BOUNCE_TIMEOUT (in module *pyAbacus.constants*), 7

C

ChecksumError, 7
 close() (in module *pyAbacus.core*), 4
 COINCIDENCE_WINDOW_DEFAULT_VALUE (in module *pyAbacus.constants*), 7
 COINCIDENCE_WINDOW_MAXIMUM_VALUE (in module *pyAbacus.constants*), 7
 COINCIDENCE_WINDOW_MINIMUM_VALUE (in module *pyAbacus.constants*), 7
 COINCIDENCE_WINDOW_STEP_VALUE (in module *pyAbacus.constants*), 7
 COUNTERS_VALUES (in module *pyAbacus.constants*), 7
 CountersValues (class in *pyAbacus.core*), 3
 CURRENT_OS (in module *pyAbacus.constants*), 7

D

dataArraysToCounters() (in module *pyAbacus.core*), 4
 dataArraysToSettings() (in module *pyAbacus.core*), 4
 dataStreamToDataArrays() (in module *pyAbacus.core*), 5
 DELAY_DEFAULT_VALUE (in module *pyAbacus.constants*), 7
 DELAY_MAXIMUM_VALUE (in module *pyAbacus.constants*), 7
 DELAY_MINIMUM_VALUE (in module *pyAbacus.constants*), 8
 DELAY_STEP_VALUE (in module *pyAbacus.constants*), 8

E

END_COMMUNICATION (in module *pyAbacus.constants*), 8
 exponentRepresentationToValue() (*pyAbacus.core.Settings48Ch* method), 4
 exponentsToBits() (*pyAbacus.core.Settings48Ch* method), 4

F

findDevices() (in module *pyAbacus.core*), 5
 findIdn() (*pyAbacus.core.AbacusSerial* method), 3
 flush() (*pyAbacus.core.AbacusSerial* method), 3
 fromBitsToValue() (*pyAbacus.core.Settings48Ch* method), 4

G

getAddressAndValue() (*pyAbacus.core.Settings2Ch* method), 3
 getAddressAndValue() (*pyAbacus.core.Settings48Ch* method), 4
 getAllCounters() (in module *pyAbacus.core*), 5
 getAllSettings() (in module *pyAbacus.core*), 5
 getChannels() (*pyAbacus.core.Settings48Ch* method), 4
 getChannelsFromName() (in module *pyAbacus.core*), 5
 getCountersID() (in module *pyAbacus.core*), 6
 getCountersID() (*pyAbacus.core.CountersValues* method), 3
 getFollowingCounters() (in module *pyAbacus.core*), 6
 getIdn() (in module *pyAbacus.core*), 6
 getIdn() (*pyAbacus.core.AbacusSerial* method), 3
 getNChannels() (*pyAbacus.core.AbacusSerial* method), 3
 getNumericAddresses() (*pyAbacus.core.CountersValues* method), 3
 getSetting() (in module *pyAbacus.core*), 6
 getSetting() (*pyAbacus.core.Settings2Ch* method), 3
 getSetting() (*pyAbacus.core.Settings48Ch* method), 4

`getSettingStr()` (*pyAbacus.core.Settings2Ch method*), 3
`getSettingStr()` (*pyAbacus.core.Settings48Ch method*), 4
`getTimeLeft()` (*in module pyAbacus.core*), 6
`getTimeLeft()` (*pyAbacus.core.CountersValues method*), 3
`getValue()` (*pyAbacus.core.CountersValues method*), 3
`getValues()` (*pyAbacus.core.CountersValues method*), 3
`getValuesFormatted()` (*pyAbacus.core.CountersValues method*), 3

I

`initAddreses()` (*pyAbacus.core.Settings48Ch method*), 4
`InvalidValueError`, 7

M

`MAXIMUM_WRITING_TRIES` (*in module pyAbacus.constants*), 8
`module`
 pyAbacus.constants, 7
 pyAbacus.core, 3
 pyAbacus.exceptions, 7

O

`open()` (*in module pyAbacus.core*), 6

P

`pyAbacus.constants`
 module, 7
`pyAbacus.core`
 module, 3
`pyAbacus.exceptions`
 module, 7

R

`READ_VALUE` (*in module pyAbacus.constants*), 8
`readSerial()` (*in module pyAbacus.core*), 6
`readSerial()` (*pyAbacus.core.AbacusSerial method*), 3
`renameDuplicates()` (*in module pyAbacus.core*), 6

S

`SAMPLING_DEFAULT_VALUE` (*in module pyAbacus.constants*), 8
`SAMPLING_VALUES` (*in module pyAbacus.constants*), 8
`setAllSettings()` (*in module pyAbacus.core*), 6
`setCounters()` (*pyAbacus.core.Stream method*), 4
`setCountersID()` (*pyAbacus.core.CountersValues method*), 3

`setSetting()` (*in module pyAbacus.core*), 6
`setSetting()` (*pyAbacus.core.Settings2Ch method*), 3
`setSetting()` (*pyAbacus.core.Settings48Ch method*), 4
`setTimeLeft()` (*pyAbacus.core.CountersValues method*), 3
`SETTINGS` (*in module pyAbacus.constants*), 8
`Settings2Ch` (*class in pyAbacus.core*), 3
`Settings48Ch` (*class in pyAbacus.core*), 4
`Settings4Ch` (*class in pyAbacus.core*), 4
`Settings8Ch` (*class in pyAbacus.core*), 4
`setValueFromArray()` (*pyAbacus.core.CountersValues method*), 3
`SLEEP_DEFAULT_VALUE` (*in module pyAbacus.constants*), 8
`SLEEP_MAXIMUM_VALUE` (*in module pyAbacus.constants*), 8
`SLEEP_MINIMUM_VALUE` (*in module pyAbacus.constants*), 8
`SLEEP_STEP_VALUE` (*in module pyAbacus.constants*), 8
`start()` (*pyAbacus.core.Stream method*), 4
`START_COMMUNICATION` (*in module pyAbacus.constants*), 8
`stop()` (*pyAbacus.core.Stream method*), 4
`Stream` (*class in pyAbacus.core*), 4

T

`testDevice()` (*pyAbacus.core.AbacusSerial method*), 3
`time_left` (*pyAbacus.core.CountersValues attribute*), 3
`TIMEOUT` (*in module pyAbacus.constants*), 8
`TimeOutError`, 7

V

`valueToExponentRepresentation()` (*pyAbacus.core.Settings48Ch method*), 4

W

`WRITE_VALUE` (*in module pyAbacus.constants*), 8
`writeSerial()` (*in module pyAbacus.core*), 7
`writeSerial()` (*pyAbacus.core.AbacusSerial method*), 3