PyAbacus Documentation

Release 1.2.1

Tausand Electronics

Jun 11, 2024

CONTENTS

pyAbacus was built to simplify the usage of Tausand Abacus family of coincidence counters, providing a library aimed to interface these devices using Python coding.

CHAPTER

ONE

CONTENTS

1.1 pyAbacus.core

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

findIdn()
Requests the device for its string identificator (IDN) using serial port.

flush()

getFamily()
Gets the family of the device: "AB1000" or "AB2000".

getIdn()
Gets the device string identificator (IDN) from local memory.

getNChannels()
Gets the number of input channels in the device.

getResolution()
Gets the resolution of the device, in nanoseconds.

readSerial()

testDevice()

writeSerial(*command*, *address*, *data_16o32*)

class pyAbacus.core.**CountersValues**(*n_channels*, *family='AB1000'*)
Keeps a set of measurements from counters within a device.

getCountersID()
Gets the *counters_id* (consecutive number of measurements) field from a set of measurements.

getNumericAddresses()

getTimeLeft()
Gets the *time_left* (time in ms for next measurement to be available) field from a set of measurements.

getValue(*channel*)
Gets a value of a single channel.

Example: mycounters.getValue('A')

Args: channel: upper case characters indicating the channel to be read. e.g. 'A' for singles in input A, 'AB' for coincidences between inputs A and B.

Returns: integer value of counts in the selected channel

getValues(*channels*)
Gets an array of values of several channels.

Example: mycounters.getValues({'A','B','AB'})

Args: channels: list of upper case characters indicating the channel to be read. e.g. 'A' for singles in input A, 'AB' for coincidences between inputs A and B.

Returns: array of integer values of counts in the selected channels

getValuesFormatted(*channels*)

setCountersID(*id*)

setTimeLeft(*time*)

setValueFromArray(*address*, *value*)

time_left
in ms

class pyAbacus.core.**Settings2Ch**

getAddressAndValue(*timer*)

getSetting(*timer*)

getSettingStr(*timer*)

setSetting(*setting*, *value*)

class pyAbacus.core.**Settings2ChAB2000**
2 channel devices of family AB2000, similar to 4 and 8 channel devices, use as time base a second. On the other hand, 2 channel devices of family AB1000 use ns for all timers with the exception of the sampling time (ms).

class pyAbacus.core.**Settings48Ch**
4 and 8 channel devices use as time base a second. Nevertheless 2 channel of family AB1000 use 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*)

initAddreses()

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. On the other hand, 2 channel devices of family AB1000 use 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. On the other hand, 2 channel devices of family AB1000 use 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*, *chunck_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.

chunck_size : integer, number of bytes per single data row.
- Use chunck_size=3 for devices with inner 16-bit registers e.g. Tausand Abacus AB1002, where byte streams are: {address,MSB,LSB}.
- Use chunck_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 A boolean value that is False if incoming data is corrupt and True otherwise

Raises: AbacusError: Input string is not valid chunck 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 inlcuded in the final answer. The constant DEVICES is updated with the dictionary of valid devices.

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

Returns: ports, len(ports) List of valid ports, and its length. ports is a dictionary where the keys are the identifier strings of the devices (e.g. "Tausand Abacus AB1004"), and the values are the corresponding pyserial port (e.g. 'COM8', or '/dev/ttyACM0').

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. If communication with the device is lost and cannot be inmediatly recovered, the private function __tryReadingDataFromDevice() will throw an UnboundLocalError.

Example:

counters, counters_id = getAllCounters('COM3')

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

counters = COUNTERS VALUES: 37
A: 1023
B: 1038
AB: 201

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: CountersValues class object including 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. If communication with the device is lost and cannot be inmediatly recovered, the private function __tryReadingDataFromDevice() will throw an UnboundLocalError.

Example: settings = getAllSettings('COM3')

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

delay_A (ns): 0
delay_B (ns): 20
sleep_A (ns): 0
sleep_B (ns): 0
coincidence_window (ns): 10
sampling (ms): 1300

Args: abacus_port: device port.

Returns: Settings2ch, Settings2chAB2000, Settings4ch or Settings8ch class object including all setting values as registered within the device.

pyAbacus.core.**getChannelsFromName**(*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.**getCommunicationStatus**()
Returns the devices communication status

Returns: True if the communication was succesfully opened or False if the connection is lost.

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*.

If communication with the device is lost and cannot be inmediatly recovered, the private function __tryReading-DataFromDevice() will throw an UnboundLocalError.

Args: abacus_port: device port.

Returns: integer, *counters_id* value.

pyAbacus.core.**getFamilyFromName**(*name*)
Returns the family number by reading the device name.

For example, if name="Tausand Abacus AB1004", returns AB1000. For example, if name="Tausand Abacus AB2502", returns AB2000.

Args: name: idn string of the device.

Returns: string, family name of device (AB1000, AB2000).

Raises: AbacusError: Not a valid abacus.

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.**getLogfilePath**()
Gets the path of log information if it has been set before

pyAbacus.core.**getPhysicalPort**(*abacus_port*)
Reads the physical port at the specified serial port.

pyAbacus.core.**getResolutionFromName**(*name*)
Returns the device resolution, in nanoseconds, by reading the device name.

For example, if name="Tausand Abacus AB1004", a 5ns device, returns 5. For example, if name="Tausand Abacus AB1504", a 2ns device, returns 2.

Args: name: idn string of the device.

Returns: integer, number of input channels in device.

Raises: AbacusError: Not a valid abacus.

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.**getStatusMessage**()
Returns a string with the connection status of the device. This is used by other Tausand software products.

pyAbacus.core.**getTimeLeft**(*abacus_port*)
Reads the remaining time for the next measurement to be ready, in ms. If communication with the device is lost and cannot be inmediatly recovered, the private function __tryReadingDataFromDevice() will throw an UnboundLocalError.

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

Args: abacus_port: a string that can be either 1) the serial port name ('COMx' in Windows or '/dev/ttyxxxx' in Mac or Linux) or 2) The name and port of a previously recognized device, namely 'Tausand Abacus ABxxxx (COMx)'. For this second option, the function findDevices() should be called first.

Returns: opened_port: a string such as 'Tausand Abacus ABxxxx (COMx)'

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.**setLogfilePath**(*path*)
Sets the path to save log information

pyAbacus.core.**setSetting**(*abacus_port*, *setting*, *value*)
Sets a configuration setting within a Tausand Abacus.

Example: setSetting('COM3' , 'sampling' , 1300)

sets the sampling time to 1300 ms to a device in port 'COM3'.

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.**setStatusMessage**(*message*)
Sets the connection status of the device. . This is used by other Tausand software products

Args: message: A string with a message that might be shown to the user.

pyAbacus.core.**waitAndGetValues**(*abacus_port*, *channels*, *print_on=False*, *max_try=6*, *max_wait_s=10*)
Waits and reads a new set of valid data from a Tausand Abacus.

Example: counters, counter_id = waitAndGetValues('COM3',{'A','B','AC'})

Waits for a new set of valid data to be available, related to the sampling time of the device. Then, reads the values of counts in A, B and the coincidences of AC, of the device connected in port COM3. Returns the requested counters within an array, for example | counters = [1023,1038,201] | counter_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 C.

Args: abacus_port: device port

channels: list of upper case characters indicating the channel to be read. e.g. 'A' for singles in input A, 'AB' for coincidences between inputs A and B.

print_on: bool When True, prints information of the waiting process

max_try: positive integer number, indicating the maximum trials to recover a communication issue

max_wait_s: timeout maximum number of seconds to wait. Once this time is reached, the function ends.

Returns: counters, counters_id

Set of read data, and their corresponding ID

counters: array of integer values of counts in the selected channels

counters_id: ID (consecutive number of measurements) field from a set of measurements.

pyAbacus.core.**waitForAcquisitionComplete**(*abacus_port*, *print_on=False*, *max_try=6*, *max_wait_s=10*)
Waits for a new set of valid data to be available within a Tausand Abacus.

Args: abacus_port: device port

print_on: bool When True, prints information of the waiting process

max_try: positive integer number, indicating the maximum trials to recover a communication issue

max_wait_s: timeout maximum number of seconds to wait. Once this time is reached, the function ends.

Returns: 0 if wait has suceeded. -1 if timeout has been reached.

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_window_ns': 20, 'coincidence_window_s': 23, 'coincidence_window_us': 21, 'counts_AB_LSB': 28, 'counts_AB_MSB': 29, 'counts_A_LSB': 24, 'counts_A_MSB': 25, 'counts_B_LSB': 26, 'counts_B_MSB': 27, 'dataID': 30, 'delay_A_ms': 2, 'delay_A_ns': 0, 'delay_A_s': 3, 'delay_A_us': 1, 'delay_B_ms': 6, 'delay_B_ns': 4, 'delay_B_s': 7, 'delay_B_us': 5, 'sampling_ms': 18, 'sampling_ns': 16, 'sampling_s': 19, 'sampling_us': 17, 'sleep_A_ms': 10, 'sleep_A_ns': 8, 'sleep_A_s': 11, 'sleep_A_us': 9, 'sleep_B_ms': 14, 'sleep_B_ns': 12, 'sleep_B_s': 15, 'sleep_B_us': 13, 'time_left': 31}**
Memory addresses

pyAbacus.constants.**BAUDRATE = 115200**
Default baudrate for the serial port communication

pyAbacus.constants.**BOUNCE_TIMEOUT = 1**
Number of times a specific transmition 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, 10000, 20000, 50000, 100000, 200000, 500000, 1000000]**
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

1.4 Examples

1.4.1 abacusExample.py

```
"""abacusExample

    Several simple examples showing how to communicate with a Tausand Abacus device␣
↪using pyAbacus library.
"""
import pyAbacus as abacus
import time

print("********************")
print("pyAbacus example")
print("********************")
print("1. Find devices and establish a connection\n")
ports, n = abacus.findDevices() #required to scan devices before opening a connection

if n==0:
    print("\nNo valid devices were found. Closing.")

else:
    print("\nAvailable valid devices:")
    print(ports)
```

(continues on next page)

(continued from previous page)

```
    mydevice = list(ports.keys())[0] #get first available device

    abacus.open(mydevice)                          #open connection with device

    print("\nConnected to the following device:")
    idnstring = abacus.getIdn(mydevice)
    numchannels = abacus.getChannelsFromName(mydevice)
    resolution = abacus.getResolutionFromName(mydevice)
    physicalport = abacus.getPhysicalPort(mydevice)
    print("   device name =",mydevice)
    print("   device physical port =",physicalport)
    print("   device identifier string =",idnstring)
    print("   number of channels =",numchannels)
    print("   resolution =",resolution,"ns")

    print("\n********************")
    print("2. Read device settings\n")

    #Example of reading all device settings
    settings = abacus.getAllSettings(mydevice)
    print("Settings read from device, using getAllSettings method:")
    print("   settings =",settings)

    #Examples reading single settings
    value = abacus.getSetting(mydevice,"delay_A")
    print("current delay_A=",value,"ns")
    value = abacus.getSetting(mydevice,"delay_B")
    print("current delay_B=",value,"ns")
    value = abacus.getSetting(mydevice,"sleep_A")
    print("current sleep_A=",value,"ns")
    value = abacus.getSetting(mydevice,"sleep_B")
    print("current sleep_B=",value,"ns")
    value = abacus.getSetting(mydevice,"coincidence_window")
    print("current coincidence_window=",value,"ns")
    value = abacus.getSetting(mydevice,"sampling")
    print("current sampling=",value,"ms")

    print("\n********************")
    print("3. Write device settings\n")

    #Example of writing a new setting value
    abacus.setSetting(mydevice,"sampling", 2000)    #set sampling=2000ms
    value = abacus.getSetting(mydevice,"sampling")  #read sampling
    print("current sampling=",value,"ms")

    abacus.setSetting(mydevice,"delay_B", 20)            #set delay_B=20ns
    value = abacus.getSetting(mydevice,"delay_B")
    print("current delay_B=",value,"ns")

    abacus.setSetting(mydevice,"coincidence_window", 50)            #set coincidence_
↪window=50ns
```

(continues on next page)

(continued from previous page)

```
    value = abacus.getSetting(mydevice,"coincidence_window")
    print("current coincidence_window=",value,"ns")

    print("waiting 2 seconds to complete the measurement")
    #  wait for a new set of data to be ready
    abacus.waitForAcquisitionComplete(mydevice) #new function on v1.1.1
    #  alternative to wait
    #time.sleep(2); #wait sampling time (2s) to get a valid measurement

    print("\n**********************")
    print("4. Read measurements from device\n")

    #Example of reading measurements from counters

    #  read data from device
    counters, counters_id = abacus.getAllCounters(mydevice)
    print("Measurements read from device, using getAllCounters method:")
    print("   counters_id =",counters_id)
    print("   counters =",counters)
    print("   counters type=",type(counters))
    print("   counts in A:                    counters.getValue('A')  =",counters.
↪getValue('A'))
    print("   coincidences between A and B:   counters.getValue('AB') =",counters.
↪getValue('AB'))

    print("   counts in A and B:              counters.getValues(['A','B']) =",counters.
↪getValues(['A','B']))

    #Example of multiple coincidences
    if numchannels >= 4:
        print("\n**********************")
        print("5. Working with multi-fold measurements\n")
        value = abacus.getSetting(mydevice,"config_custom_c1")
        print("Current settings for multi-channel coincidences: ",value)
        value = counters.getValue("custom_c1")
        print("Current value of multi-channel coincidences: ",value)
        print("\nNew settings for multi-channel coincidences as ABC")
        abacus.setSetting(mydevice,"config_custom_c1","ABC")#must use 3 or 4 letters.␣
↪Valid options: ABC, ABD, ACD, BCD, ABCD
        value = abacus.getSetting(mydevice,"config_custom_c1")
        print("Current settings for multi-channel coincidences: ",value)
        print("waiting 2 seconds to complete the measurement")
        #  wait for a new set of data to be ready
        abacus.waitForAcquisitionComplete(mydevice) #new function on v1.1.1
        counters, counters_id = abacus.getAllCounters(mydevice)
        value = counters.getValue("custom_c1")
        print("Current value of multi-channel coincidences: ",value)

    abacus.close(mydevice)                      #close connection with device
```

1.4.2 multipleReadExample.py

```
"""multipleReadExample

    Reads continuous samples from a Tausand Abacus device, using pyAbacus library.
    Implements error handling to self-recover.
    Writes measured data in a plain text file.
"""
import pyAbacus as abacus
import time

samples_to_read = 10 #change this parameter to set how many samples to read
my_sampling_time_ms = 1000 #change this parameter to set your sampling time. 1000=1s.

#define the desired channels to be read. Some examples:
channels_to_read_2ch = ['A','B','AB']   #for a 2 channel device
channels_to_read_4ch = ['A','B','C','AB','AC','custom_c1'] #custom_c1 corresponds to a␣
↪multi-fold mesurement, to be configured, e.g. 'ABC'
channels_to_read_8ch = ['A','B','C','AB','AC','custom_c1', 'custom_c3', 'custom_c8']
↪#custom_cN corresponds to a multi-fold mesurement, to be configured, e.g. 'ABC'

print("******************************")
print("pyAbacus multiple read example")
print("******************************")
print("1. Find devices and establish a connection\n")

ports, n = abacus.findDevices() #required to scan devices before opening a connection

if n==0:
    print("\nNo valid devices were found. Closing.")

else:
    print("\nAvailable valid devices:")
    print(ports)

    mydevice = list(ports.keys())[0] #get first available device
    abacus.open(mydevice)             #open connection with device

    #get and print some properties of the connected device:
    print("\nConnected to the following device:")
    idnstring = abacus.getIdn(mydevice)
    numchannels = abacus.getChannelsFromName(mydevice)
    resolution = abacus.getResolutionFromName(mydevice)
    physicalport = abacus.getPhysicalPort(mydevice)
    print("   device name =",mydevice)
    print("   device physical port =",physicalport)
    print("   device identifier string =",idnstring)
    print("   number of channels =",numchannels)
    print("   resolution =",resolution,"ns")

    ## select channels_to_read depending on the number of channels of the connected␣
↪device
    if numchannels == 8:
```

(continues on next page)

(continued from previous page)

```
            channels_to_read = channels_to_read_8ch
    elif numchannels == 4:
        channels_to_read = channels_to_read_4ch
    else:
        channels_to_read = channels_to_read_2ch

    #########
    print("\n2. Write and read new settings\n")
    #write settings, using pyAbacus setSetting function:
    abacus.setSetting(mydevice,"sampling", my_sampling_time_ms) #set sampling, in␣
↪milliseconds (default: 1000ms)
    abacus.setSetting(mydevice,"coincidence_window", 50)    #set coincidence_window=50ns
    abacus.setSetting(mydevice,"delay_A", 0)                    #set delay_A=0ns
    abacus.setSetting(mydevice,"delay_B", 0)                    #set delay_B=0ns
    abacus.setSetting(mydevice,"sleep_A", 0)                    #set sleep_A=0ns
    abacus.setSetting(mydevice,"sleep_B", 0)                    #set sleep_B=0ns
    if numchannels == 4:
        abacus.setSetting(mydevice,"delay_C", 0)                #set delay_C=0ns
        abacus.setSetting(mydevice,"sleep_C", 0)                #set sleep_C=0ns
        #configure multi-fold coincidences ('custom_c1'):
        abacus.setSetting(mydevice,"config_custom_c1","ABC")    #must use 3 or 4 letters.
↪ Valid options: ABC, ABD, ACD, BCD, ABCD
    elif numchannels == 8:
            abacus.setSetting(mydevice,"config_custom_c1","ADE")        #must use 3 or 4␣
↪letters. Valid options: any combination of letters from A to H in alphabetical order
            abacus.setSetting(mydevice,"config_custom_c3","EFH")
            abacus.setSetting(mydevice,"config_custom_c8","BEGH")

    #read settings, using pyAbacus getAllSettings function:
    #a retry routine has been implemented, as an example to handle exceptions
    max_attempts=5
    for attempt in range(max_attempts):
        try:
            current_settings = abacus.getAllSettings(mydevice)
            print(current_settings)
        except abacus.CheckSumError:
            print("Data integrity error in getAllSettings: missing or wrong bits. Retry.
↪")
            pass #retry
        except (abacus.serial.serialutil.SerialException,KeyError):
            print("Communication error in getAllSettings: device might be disconnected␣
↪or turned off.")
        except:
            print("Unexpected error. Device connection closed.")
            abacus.close(mydevice)  #close connection with device
            raise
        else:
            break #done, continue



    #########
```

(continues on next page)

(continued from previous page)

```
    print("\n3. Create file")
    #define constant strings:
    date_time_string = time.strftime("%Y-%m-%d_%H%M%S", time.localtime())
    column_headers = ["PC time","countersID"]+channels_to_read;
    file_name = "data_multipleReadExample_"+date_time_string+".txt"
    #create file and write headers:
    myfile = open(file_name, "a")
    print("File",file_name,"has been created")
    myfile.write('multipleReadExample\n')
    myfile.write('--------------------\n')
    myfile.write('Begin time: '+date_time_string+'\n')
    myfile.write('Device: '+mydevice+'\n')
    myfile.write('Settings: '+str(current_settings)+'\n')
    myfile.write("\n\n")
    myfile.write(str(column_headers))
    myfile.write("\n")

    #########
    print("\n4. Multiple read using pyAbacus waitAndGetValues function begins")
    full_data=[]
    print(column_headers)
    for sample in range(samples_to_read):
        print(sample+1,'/',samples_to_read,sep='',end=',')
        try:
            my_data, my_id = abacus.waitAndGetValues(mydevice,channels_to_read,print_
↪on=True) #for debug purposes, turn print_on=True; by default is False.
            my_data.insert(0,my_id) #prepend my_id to my_data
            my_data.insert(0,round(time.time(),3)) #prepend current PC time to my_data
            print(my_data)
            full_data.append(my_data)
            myfile.write(str(my_data))
            myfile.write("\n")
        except abacus.CheckSumError:
            print("Data integrity error in waitAndGetValues: missing or wrong bits.")
        except (abacus.serial.serialutil.SerialException,KeyError):
            print("Communication error in waitAndGetValues: device might be disconnected␣
↪or turned off.")
        except:
            print("Unexpected error. Device connection closed. File access closed.")
            abacus.close(mydevice)  #close connection with device
            myfile.close()          #close file
            raise

    abacus.close(mydevice)  #close connection with device
    myfile.close()          #close file
    print("\nFull set of data is:")
    print(column_headers)
    print(full_data)
    print("\nExample done.")
```

1.4.3 openPortExample.py

```
"""openPortExample

    Show how to start communication with a Tausand Abacus devices using two simple␣
↪methods: 1) By specifying the port name a device is plugged into.    2) By finding␣
↪Tausand Abacus devices that are plugged in and connecting to the first found device
    Finally, after comminucation is established, the counters values are read.
"""
import pyAbacus as abacus

# FIRST METHOD
print('\nFIRST METHOD\n')
# The simplest way to start communication with a Tausand device is by calling the␣
↪function open, assuming you know what port your device is plugged into:
#my_tausand = abacus.open("COM3") # For Windows use a string "COMX" where X is the␣
↪number of the port
my_tausand = abacus.open("/dev/ttyACM0") # For Mac or Linux use "/dev/ttyXXXXXXX", where␣
↪ttyXXXXXXX represents the name of the port
data = abacus.getAllCounters(my_tausand)
abacus.close(my_tausand)
print(data)

# SECOND METHOD
print('\nSECOND METHOD\n')
# If you don't know the ports' names, you can also start communication by calling find␣
↪devices(), choosing one of the ports, and calling open():
ports , n = abacus.findDevices()    #required to scan devices before opening a connection
my_tausand = list(ports.keys())[0]  #get first available device
abacus.open(my_tausand)             #open connection with device
data = abacus.getAllCounters(my_tausand)    #read data
abacus.close(my_tausand)            #close connection with device
print(data)
```

1.4.4 readSpeedTestExample.py

```
# -*- coding: utf-8 -*-
"""readSpeedTestExample

    Repeat reading of counters and settings for a specified number of times, and returns␣
↪statistics on the timing of execution of these reading functions:
    * getAllCounters()
    * getAllSettings()

    Usage:
       Set the port variable in the corresponding value.
       Run.

    Created on Mon Jan 23 16:57:32 2023
"""
```

(continues on next page)

(continued from previous page)

```
import pyAbacus as abacus
from time import time
import numpy as np

##User's parameters
port = 'COM4'    #indicate the port to connect with. E.g.: 'COM4'
samples = 1000   #how many times the read test should be made

##
#create empty arrays
tRdCounters=[]
tRdSettings=[]

my_tausand = abacus.open(port);             #open connection with device

print("Connected to: ",end='')
print(abacus.getIdn(my_tausand));

print("Read speed test. Progress=%5.1f" % ((0.0)), end='')
for a in range(samples):
    t0 = time() #start timer
    data = abacus.getAllCounters(my_tausand)    #read data
    tf = time() #end timer
    tRdCounters.append(tf-t0) #write lapsed time

    t0 = time() #start timer
    settings = abacus.getAllSettings(my_tausand)    #read settings
    tf = time() #end timer
    tRdSettings.append(tf-t0) #write lapsed time

    progress = float((a+1)*100.0/samples)
    print("\b\b\b\b\b", end='')
    print("%5.1f" % (progress), end='')

print("")
abacus.close(my_tausand)            #close connection with device

print("NumReads:  %d" % (a+1))
print("getAllCounters() statistics")
print("  Min:  %0.5f s" % (np.min(tRdCounters)))
print("  Max:  %0.5f s" % (np.max(tRdCounters)))
print("  Mean: %0.5f s" % (np.mean(tRdCounters)))
print("  StdD: %0.5f s" % (np.std(tRdCounters)))
print("getAllSettings() statistics")
print("  Min:  %0.5f s" % (np.min(tRdSettings)))
print("  Max:  %0.5f s" % (np.max(tRdSettings)))
print("  Mean: %0.5f s" % (np.mean(tRdSettings)))
print("  StdD: %0.5f s" % (np.std(tRdSettings)))
```

1.4.5 singleReadExample.py

```
"""singleReadExample

    Finds Tausand Abacus devices, connects to the first found device, and reads counter␣
↪values from device.
"""
import pyAbacus as abacus
ports , n = abacus.findDevices()    #required to scan devices before opening a connection
my_tausand = list(ports.keys())[0]  #get first available device
abacus.open(my_tausand)             #open connection with device
data = abacus.getAllCounters(my_tausand)    #read data
abacus.close(my_tausand)            #close connection with device
print(data)
```

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX