pyModbusTCP documentation

version 0.0.13

Loic Lefebvre

May 30, 2016

Contents

Welcome to pyModbusTCP's documentation **1**

Quick start guide 1

Overview of the package 1

Package setup 1

ModbusClient: init 1

ModbusClient: manage TCP link 1

ModbusClient: available modbus requests functions 2

ModbusClient: debug mode 2

utils module: Modbus data mangling 3

pyModbusTCP modules documentation 4

Module pyModbusTCP.client 4

class pyModbusTCP.client.ModbusClient 4

Module pyModbusTCP.utils 7

pyModbusTCP examples 8

Simple read registers example 8

Simple read bits example 9

Simple write bits example 10

An example with a modbus polling thread 11

Indices and tables **12**

Index **13**

Python Module Index **15**

Welcome to pyModbusTCP's documentation

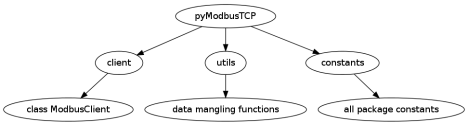
Quick start guide

Overview of the package

pyModbusTCP give access to modbus/TCP server through the ModbusClient object. This class is define in the client module.

To deal with frequent need of modbus data mangling (for example 32 bits IEEE float to 2x16 bits words convertion) a special module named utils provide some helful functions.

Package map:



Package setup

from PyPi:

```
# for Python 2.7
sudo pip-2.7 install pyModbusTCP
# or for Python 3.2
sudo pip-3.2 install pyModbusTCP
# or upgrade from an older release
sudo pip-3.2 install pyModbusTCP --upgrade
```

from Github:

```
git clone https://github.com/sourceperl/pyModbusTCP.git
cd pyModbusTCP
# here change "python" by your python target(s) version(s) (like python3.2)
sudo python setup.py install
```

ModbusClient: init

init module from constructor (raise ValueError if host/port error):

```
from pyModbusTCP.client import ModbusClient
try:
    c = ModbusClient(host="localhost", port=502)
except ValueError:
    print("Error with host or port params")
```

you can also init module from functions host/port return None if error:

```
from pyModbusTCP.client import ModbusClient
c = ModbusClient()
if not c.host("localhost"):
    print("host error")
if not c.port(502):
    print("port error")
```

ModbusClient: manage TCP link

Now, it's possible to use auto mode to let module deal with TCP open/close.

For keep TCP open, add auto_open=True in init:

```
c = ModbusClient(host="localhost", auto_open=True)
```

For open/close socket before/after read or write, do this:

```
c = ModbusClient(host="localhost", auto_open=True, auto_close=True)
```

You can also open manually the TCP link. After this, you call a modbus request function (see list in next section):

```
if c.open():
    regs_list_1 = c.read_holding_registers(0, 10)
    regs_list_2 = c.read_holding_registers(55, 10)
    c.close()
```

With a forever polling loop, TCP always open (auto-reconnect code):

```
while True:
    if c.is_open():
        regs_list_1 = c.read_holding_registers(0, 10)
        regs_list_2 = c.read_holding_registers(55, 10)
    else:
        c.open()
    time.sleep(1)
```

ModbusClient: available modbus requests functions

See http://en.wikipedia.org/wiki/Modbus for full table.

Domain	Function name	Function code	ModbusClient function
Bit	Read Discrete Inputs	2	**read_discrete_inputs()**
	Read Coils	1	**read_coils()**
	Write Single Coil	5	**write_single_coil()**
	Write Multiple Coils	15	**write_multiple_coils()**
Register	Read Input Registers	4	**read_input_registers()**
	Read Holding Registers	3	**read_holding_registers()**
	Write Single Register	6	**write_single_register()**
	Write Multiple Registers	16	**write_multiple_registers()**
	Read/Write Multiple Registers	23	n/a
	Mask Write Register	22	n/a
File	Read FIFO Queue	24	n/a
	Read File Record	20	n/a
	Write File Record	21	n/a
	Read Exception Status	7	n/a
Diagnostic	Diagnostic	8	n/a
	Get Com Event Counter	11	n/a
	Get Com Event Log	12	n/a
	Report Slave ID	17	n/a
	Read Device Identification	43	n/a

ModbusClient: debug mode

If need, you can enable a debug mode for ModbusClient like this:

```
from pyModbusTCP.client import ModbusClient
c = ModbusClient(host="localhost", port=502, debug=True)
```

or:

```
c.debug(True)
```

when debug is enable all debug message is print on console and you can see modbus frame:

```
c.read_holding_registers(0, 4)
```

print:

```
Tx
[E7 53 00 00 00 06 01] 03 00 00 00 04
Rx
[E7 53 00 00 00 0B 01] 03 08 00 00 00 6F 00 00 00 00
[0, 111, 0, 0]
```

utils module: Modbus data mangling

Sample data mangling, usefull for interface PLC device.

- 16 bits to 32 bits integers:

```
from pyModbusTCP import utils
list_16_bits = [0x0123, 0x4567, 0x89ab, 0xcdef]

# big endian sample (default)
list_32_bits = utils.word_list_to_long(list_16_bits)
# display "['0x1234567', '0x89abcdef']"
print([hex(i) for i in list_32_bits])

# little endian sample
list_32_bits = utils.word_list_to_long(list_16_bits, big_endian=False)
# display "['0x45670123', '0xcdef89ab']"
print([hex(i) for i in list_32_bits])
```

- two's complement (see http://en.wikipedia.org/wiki/Two%27s_complement):

```
from pyModbusTCP import utils
list_16_bits = [0x0000, 0xFFFF, 0x00FF, 0x8001]

# display "[0, -1, 255, -32767]"
print(utils.get_list_2comp(list_16_bits, 16))

# display "-1"
print(utils.get_2comp(list_16_bits[1], 16))
```

- an integer of val_size bits (default is 16) to an array of boolean:

```
from pyModbusTCP import utils
# display "[True, False, True, False, False, False, False, False]"
print(utils.get_bits_from_int(0x05, val_size=8))
```

- gateway between IEEE single precision float and python float:

```
from pyModbusTCP import utils

# convert python float 0.3 to 0x3e99999a (32 bits IEEE representation)
# display "0x3e99999a"
print(hex(utils.encode_ieee(0.3)))

# convert python float 0.3 to 0x3e99999a (32 bits IEEE representation)
# display "0.300000011921" (it's not 0.3, precision leak with float...)
print(utils.decode_ieee(0x3e99999a))
```

pyModbusTCP modules documentation

Contents:

Module pyModbusTCP.client

This module provide the ModbusClient class used to deal with modbus server.

class pyModbusTCP.client.ModbusClient

class pyModbusTCP.client.**ModbusClient** (*host=None*, *port=None*, *unit_id=None*, *timeout=None*, *debug=None*, *auto_open=None*, *auto_close=None*)

Client Modbus TCP

__init__ (*host=None*, *port=None*, *unit_id=None*, *timeout=None*, *debug=None*, *auto_open=None*, *auto_close=None*)

Constructor

Modbus server params (host, port) can be set here or with host(), port() functions. Same for debug option.
Use functions avoid to launch ValueError except if params is incorrect.

Parameters:

- **host** (*str*) -- hostname or IPv4/IPv6 address server address (optional)
- **port** (*int*) -- TCP port number (optional)
- **unit_id** (*int*) -- unit ID (optional)
- **timeout** (*float*) -- socket timeout in seconds (optional)
- **debug** (*bool*) -- debug state (optional)
- **auto_open** (*bool*) -- auto TCP connect (optional)
- **auto_close** (*bool*) -- auto TCP close (optional)

Returns: Object ModbusClient

Return type: ModbusClient

Raises ValueError: if a set parameter value is incorrect

auto_close (*state=None*)

Get or set automatic TCP close mode (after each request)

Parameters: **state** (*bool or None*) -- auto_close state or None for get value

Returns: auto_close state or None if set fail

Return type: bool or None

auto_open (*state=None*)

Get or set automatic TCP connect mode

Parameters: **state** (*bool or None*) -- auto_open state or None for get value

Returns: auto_open state or None if set fail

Return type: bool or None

close ()

Close TCP connection

Returns: close status (True for close/None if already close)

Return type: bool or None

debug (*state=None*)

Get or set debug mode

Parameters: **state** (*bool or None*) -- debug state or None for get value
Returns: debug state or None if set fail
Return type: bool or None

host (*hostname=None*)
Get or set host (IPv4/IPv6 or hostname like 'plc.domain.net')

Parameters: **hostname** (*str or None*) -- hostname or IPv4/IPv6 address or None for get value
Returns: hostname or None if set fail
Return type: str or None

is_open ()
Get status of TCP connection

Returns: status (True for open)
Return type: bool

last_error ()
Get last error code

Returns: last error code
Return type: int

last_except ()
Get last except code

Returns: last except code
Return type: int

mode (*mode=None*)
Get or set modbus mode (TCP or RTU)

Parameters: **mode** (*int*) -- mode (MODBUS_TCP/MODBUS_RTU) to set or None for get value
Returns: mode or None if set fail
Return type: int or None

open ()
Connect to modbus server (open TCP connection)

Returns: connect status (True if open)
Return type: bool

port (*port=None*)
Get or set TCP port

Parameters: **port** (*int or None*) -- TCP port number or None for get value
Returns: TCP port or None if set fail
Return type: int or None

read_coils (*bit_addr*, *bit_nb=1*)
Modbus function READ_COILS (0x01)

Parameters:
- **bit_addr** (*int*) -- bit address (0 to 65535)
- **bit_nb** (*int*) -- number of bits to read (1 to 2000)

Returns: bits list or None if error
Return type: list of bool or None

read_discrete_inputs (*bit_addr*, *bit_nb=1*)

Modbus function READ_DISCRETE_INPUTS (0x02)

Parameters:
- **bit_addr** (*int*) -- bit address (0 to 65535)
- **bit_nb** (*int*) -- number of bits to read (1 to 2000)

Returns: bits list or None if error

Return type: list of bool or None

read_holding_registers (*reg_addr*, *reg_nb=1*)

Modbus function READ_HOLDING_REGISTERS (0x03)

Parameters:
- **reg_addr** (*int*) -- register address (0 to 65535)
- **reg_nb** (*int*) -- number of registers to read (1 to 125)

Returns: registers list or None if fail

Return type: list of int or None

read_input_registers (*reg_addr*, *reg_nb=1*)

Modbus function READ_INPUT_REGISTERS (0x04)

Parameters:
- **reg_addr** (*int*) -- register address (0 to 65535)
- **reg_nb** (*int*) -- number of registers to read (1 to 125)

Returns: registers list or None if fail

Return type: list of int or None

timeout (*timeout=None*)

Get or set timeout field

Parameters: **timeout** (*float or None*) -- socket timeout in seconds or None for get value

Returns: timeout or None if set fail

Return type: float or None

unit_id (*unit_id=None*)

Get or set unit ID field

Parameters: **unit_id** (*int or None*) -- unit ID (0 to 255) or None for get value

Returns: unit ID or None if set fail

Return type: int or None

version ()

Get package version

Returns: current version of the package (like "0.0.1")

Return type: str

write_multiple_coils (*bits_addr*, *bits_value*)

Modbus function WRITE_MULTIPLE_COILS (0x0F)

Parameters:
- **bits_addr** (*int*) -- bits address (0 to 65535)
- **bits_value** (*list*) -- bits values to write

Returns: True if write ok or None if fail

Return type: bool or None

write_multiple_registers (*regs_addr*, *regs_value*)

Modbus function WRITE_MULTIPLE_REGISTERS (0x10)

Parameters:
- **regs_addr** (*int*) -- registers address (0 to 65535)
- **regs_value** (*list*) -- registers values to write

Returns: True if write ok or None if fail

Return type: bool or None

write_single_coil (*bit_addr*, *bit_value*)
Modbus function WRITE_SINGLE_COIL (0x05)

Parameters:
- **bit_addr** (*int*) -- bit address (0 to 65535)
- **bit_value** (*bool*) -- bit value to write

Returns: True if write ok or None if fail

Return type: bool or None

write_single_register (*reg_addr*, *reg_value*)
Modbus function WRITE_SINGLE_REGISTER (0x06)

Parameters:
- **reg_addr** (*int*) -- register address (0 to 65535)
- **reg_value** (*int*) -- register value to write

Returns: True if write ok or None if fail

Return type: bool or None

Module pyModbusTCP.utils

This module provide a set of functions for modbus data mangling.

pyModbusTCP.utils.**crc16** (*frame*)
Compute CRC16

Parameters: **frame** (*str (Python2) or class bytes (Python3)*) -- frame

Returns: CRC16

Return type: int

pyModbusTCP.utils.**decode_ieee** (*val_int*)
Decode Python int (32 bits integer) as an IEEE single precision format
Support NaN.

Parameters: **val_int** (*int*) -- a 32 bit integer as an int Python value

Returns: float result

Return type: float

pyModbusTCP.utils.**encode_ieee** (*val_float*)
Encode Python float to int (32 bits integer) as an IEEE single precision
Support NaN.

Parameters: **val_float** (*float*) -- float value to convert

Returns: IEEE 32 bits (single precision) as Python int

Return type: int

pyModbusTCP.utils.**get_2comp** (*val_int*, *val_size=16*)
Get the 2's complement of Python int val_int

Parameters:
- **val_int** (*int*) -- int value to apply 2's complement
- **val_size** (*int*) -- bit size of int value (word = 16, long = 32) (optional)

Returns: 2's complement result

Return type: int

pyModbusTCP.utils.**get_bits_from_int** (*val_int*, *val_size=16*)

Get the list of bits of val_int integer (default size is 16 bits)
Return bits list, least significant bit first. Use list.reverse() if need.

Parameters:
- **val_int** (*int*) -- integer value
- **val_size** (*int*) -- bit size of integer (word = 16, long = 32) (optional)

Returns: list of boolean "bits" (least significant first)

Return type: list

`pyModbusTCP.utils.`**`get_list_2comp`** (*`val_list`*, *`val_size=16`*)
Get the 2's complement of Python list val_list

Parameters:
- **val_list** (*list*) -- list of int value to apply 2's complement
- **val_size** (*int*) -- bit size of int value (word = 16, long = 32) (optional)

Returns: 2's complement result

Return type: list

`pyModbusTCP.utils.`**`reset_bit`** (*`value`*, *`offset`*)
Reset a bit at offset position

Parameters:
- **value** (*int*) -- value of integer where reset the bit
- **offset** (*int*) -- bit offset (0 is lsb)

Returns: value of integer with bit reset

Return type: int

`pyModbusTCP.utils.`**`set_bit`** (*`value`*, *`offset`*)
Set a bit at offset position

Parameters:
- **value** (*int*) -- value of integer where set the bit
- **offset** (*int*) -- bit offset (0 is lsb)

Returns: value of integer with bit set

Return type: int

`pyModbusTCP.utils.`**`test_bit`** (*`value`*, *`offset`*)
Test a bit at offset position

Parameters:
- **value** (*int*) -- value of integer to test
- **offset** (*int*) -- bit offset (0 is lsb)

Returns: value of bit at offset position

Return type: bool

`pyModbusTCP.utils.`**`word_list_to_long`** (*`val_list`*, *`big_endian=True`*)
Word list (16 bits int) to long list (32 bits int)
By default word_list2long() use big endian order. For use little endian, set big_endian param to False.

Parameters:
- **val_list** (*list*) -- list of 16 bits int value
- **big_endian** (*bool*) -- True for big endian/False for little (optional)

Returns: 2's complement result

Return type: list

pyModbusTCP examples

Here some examples to see pyModbusTCP in some usages cases

Simple read registers example

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# read_register
# read 10 registers and print result on stdout

# you can use the tiny modbus server "mbserverd" to test this code
# mbserverd is here: https://github.com/sourceperl/mbserverd

# the command line modbus client mbtget can also be useful
# mbtget is here: https://github.com/sourceperl/mbtget

from pyModbusTCP.client import ModbusClient
import time

SERVER_HOST = "localhost"
SERVER_PORT = 502

c = ModbusClient()

# uncomment this line to see debug message
#c.debug(True)

# define modbus server host, port
c.host(SERVER_HOST)
c.port(SERVER_PORT)

while True:
    # open or reconnect TCP to server
    if not c.is_open():
        if not c.open():
            print("unable to connect to "+SERVER_HOST+":"+str(SERVER_PORT))

    # if open() is ok, read register (modbus function 0x03)
    if c.is_open():
        # read 10 registers at address 0, store result in regs list
        regs = c.read_holding_registers(0, 10)
        # if success display registers
        if regs:
            print("reg ad #0 to 9: "+str(regs))

    # sleep 2s before next polling
    time.sleep(2)
```

Simple read bits example

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# read_bit
# read 10 bits and print result on stdout

from pyModbusTCP.client import ModbusClient
import time

SERVER_HOST = "localhost"
SERVER_PORT = 502

c = ModbusClient()
```

```
# uncomment this line to see debug message
#c.debug(True)

# define modbus server host, port
c.host(SERVER_HOST)
c.port(SERVER_PORT)

while True:
    # open or reconnect TCP to server
    if not c.is_open():
        if not c.open():
            print("unable to connect to "+SERVER_HOST+":"+str(SERVER_PORT))

    # if open() is ok, read coils (modbus function 0x01)
    if c.is_open():
        # read 10 bits at address 0, store result in regs list
        bits = c.read_coils(0, 10)
        # if success display registers
        if bits:
            print("bit ad #0 to 9: "+str(bits))

    # sleep 2s before next polling
    time.sleep(2)
```

Simple write bits example

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# write_bit
# write 4 bits to True, wait 2s, write False, restart...

from pyModbusTCP.client import ModbusClient
import time

SERVER_HOST = "localhost"
SERVER_PORT = 502

c = ModbusClient()

# uncomment this line to see debug message
#c.debug(True)

# define modbus server host, port
c.host(SERVER_HOST)
c.port(SERVER_PORT)

toggle = True

while True:
    # open or reconnect TCP to server
    if not c.is_open():
        if not c.open():
            print("unable to connect to "+SERVER_HOST+":"+str(SERVER_PORT))

    # if open() is ok, write coils (modbus function 0x01)
    if c.is_open():
        # write 4 bits in modbus address 0 to 3
```

```
        print("")
        print("write bits")
        print("----------")
        print("")
        for addr in range(4):
            is_ok = c.write_single_coil(addr, toggle)
            if is_ok:
                print("bit #" + str(addr) + ": write to " + str(toggle))
            else:
                print("bit #" + str(addr) + ": unable to write " + str(toggle))
            time.sleep(0.5)

        time.sleep(1)

        print("")
        print("read bits")
        print("---------")
        print("")
        bits = c.read_coils(0, 4)
        if bits:
            print("bits #0 to 3: "+str(bits))
        else:
            print("unable to read")

    toggle = not toggle
    # sleep 2s before next polling
    time.sleep(2)
```

An example with a modbus polling thread

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# modbus_thread
# start a thread for polling a set of registers, display result on console
# exit with ctrl+c

import time
from threading import Thread, Lock
from pyModbusTCP.client import ModbusClient

SERVER_HOST = "localhost"
SERVER_PORT = 502

# set global
regs = []

# init a thread lock
regs_lock = Lock()

# modbus polling thread
def polling_thread():
    global regs
    c = ModbusClient(host=SERVER_HOST, port=SERVER_PORT)
    # polling loop
    while True:
        # keep TCP open
        if not c.is_open():
            c.open()
```

```
        # do modbus reading on socket
        reg_list = c.read_holding_registers(0,10)
        # if read is ok, store result in regs (with thread lock synchronization)
        if reg_list:
            with regs_lock:
                regs = reg_list
        # 1s before next polling
        time.sleep(1)

# start polling thread
tp = Thread(target=polling_thread)
# set daemon: polling thread will exit if main thread exit
tp.daemon = True
tp.start()

# display loop (in main thread)
while True:
    # print regs list (with thread lock synchronization)
    with regs_lock:
        print(regs)
    # 1s before next print
    time.sleep(1)
```

Indices and tables

- *genindex*
- *modindex*
- *search*

Index

_

__init__() (pyModbusTCP.client.ModbusClient method)

A

auto_close() (pyModbusTCP.client.ModbusClient method)

auto_open() (pyModbusTCP.client.ModbusClient method)

C

close() (pyModbusTCP.client.ModbusClient method)

crc16() (in module pyModbusTCP.utils)

D

debug() (pyModbusTCP.client.ModbusClient method)

decode_ieee() (in module pyModbusTCP.utils)

E

encode_ieee() (in module pyModbusTCP.utils)

G

get_2comp() (in module pyModbusTCP.utils)

get_bits_from_int() (in module pyModbusTCP.utils)

get_list_2comp() (in module pyModbusTCP.utils)

H

host() (pyModbusTCP.client.ModbusClient method)

I

is_open() (pyModbusTCP.client.ModbusClient method)

L

last_error() (pyModbusTCP.client.ModbusClient method)

last_except() (pyModbusTCP.client.ModbusClient method)

M

ModbusClient (class in pyModbusTCP.client)

mode() (pyModbusTCP.client.ModbusClient method)

O

open() (pyModbusTCP.client.ModbusClient method)

P

port() (pyModbusTCP.client.ModbusClient method)

pyModbusTCP.client (module)

pyModbusTCP.utils (module)

R

read_coils() (pyModbusTCP.client.ModbusClient method)

read_discrete_inputs() (pyModbusTCP.client.ModbusClient method)

read_holding_registers() (pyModbusTCP.client.ModbusClient method)

read_input_registers() (pyModbusTCP.client.ModbusClient method)

reset_bit() (in module pyModbusTCP.utils)

S

set_bit() (in module pyModbusTCP.utils)

T

test_bit() (in module pyModbusTCP.utils)

timeout() (pyModbusTCP.client.ModbusClient method)

U

unit_id() (pyModbusTCP.client.ModbusClient method)

V

version() (pyModbusTCP.client.ModbusClient method)

W

word_list_to_long() (in module pyModbusTCP.utils)

write_multiple_coils() (pyModbusTCP.client.ModbusClient method)

write_multiple_registers() (pyModbusTCP.client.ModbusClient method)

write_single_coil() (pyModbusTCP.client.ModbusClient method)

write_single_register() (pyModbusTCP.client.ModbusClient method)

Python Module Index

p

pyModbusTCP

pyModbusTCP.client

pyModbusTCP.utils