python-snap7 Documentation

Release 0.0rc0

Gijs Molenaar, Stephan Preeker

Jan 19, 2021

Contents

Contents:

CHAPTER 1

Introduction

python-snap7 is a Python wrapper for the Snap7 library. Snap7 is an open source, 32/64 bit, multi-platform Ethernet communication suite for interfacing natively with Siemens S7 PLCs.

Python-snap7 is developer for snap7 1.1.0 and Python 3.6+. It is tested on Windows (8.1 64 bit) and Linux, but it may work on other operating systems. Python Versions <3.6 may work, but is not supported anymore.

The project development is centralized on github.

CHAPTER 2

Installation

here you can find out how to install python-snap7 on your system.

2.1 Snap7

To use python-snap7 you need to have the snap7 library installed.

2.1.1 Ubuntu

If you are using Ubuntu you can use the Ubuntu packages from our launchpad PPA. To install:

```
$ sudo add-apt-repository ppa:gijzelaar/snap7
$ sudo apt-get update
$ sudo apt-get install libsnap7-1 libsnap7-dev
```

2.1.2 Windows

Download the zip file from the sourceforce page. Unzip the zip file, and copy* releaseWindows<Win64/Win32>snap7.dll* somewhere in you system PATH, for example *C:WINDOWSsystem32*. Alternatively you can copy the file somewhere on your file system and adjust the system PATH.

2.1.3 Compile from source

If you are not using Ubuntu or if you want to have more control you can download the latest source from the sourceforce page and do a manual compile. Download the file and run:

```
$ p7zip -d snap7-full-1.0.0.7z  # requires the p7 program
$ cd build/<platform>           # where platform is unix or windows
$ make -f <arch>.mk install     # where arch is your architecture, for example x86_64_
↪linux
```

(continues on next page)

(continued from previous page)

For more information about or help with compilation please check out the documentation on the snap7 website.

2.2 snap7-python

python-snap7 is available on PyPI. You can install it by using pip:

```
$ pip install python-snap7
```

You can also install it from the git repository or from a source tarball:

```
$ python ./setup.py install
```

CHAPTER 3

development

3.1 Github

We develop python-snap7 on github. If you have any problems with python-snap7 please raise an issue in the issue tracker. Even better is if you have a solution to problem! In that case you can make our live easier by following these steps:

- fork our repository on Github
- Add a tests that will fail because of the problem
- Fix the problem
- Run the test suite again
- Commit to your repository
- Issue a github pull request.

Also we try to be as much pep8 compatible as possible, where possible and reasonable.

3.2 Test suite

python-snap7 comes with a test suite with close to 100% coverage. This test suite verifies that the code actually works and makes development much easier. To run all tests please run from the source:

```
$ make test
```

Note that some tests require to run as root, since snap7 needs to bind on a privileged TCP port.

If the test complain about missing Python modules make sure the source directory is in your *PYTHONPATH* environment variable, or the python-snap7 module is installed.

3.3 Credits

python-snap7 is created by Gijs Molenaar and Stephan Preeker.

Special thanks to go to Davide Nardella for creating snap7, Thomas Hergenhahn for his libnodave and Thomas W for his S7comm wireshark plugin.

CHAPTER 4

Client

Snap7 client used for connection to a siemens7 server.

class snap7.client.**Client**

A snap7 client

ab_read(*start: int*, *size: int*) → bytearray

This is a lean function of Cli_ReadArea() to read PLC process outputs.

ab_write(*start: int*, *data: bytearray*) → int

This is a lean function of Cli_WriteArea() to write PLC process outputs

as_ab_read(*start: int*, *size: int*, *data*) → int

This is the asynchronous counterpart of client.ab_read().

as_ab_write(*start: int*, *data: bytearray*) → int

This is the asynchronous counterpart of Cli_ABWrite.

as_compress(*time: int*) → int

This is the asynchronous counterpart of client.compress().

as_db_get(*db_number: int*, *_buffer*, *size*) → bytearray

This is the asynchronous counterpart of Cli_DBGet.

as_download(*data: bytearray*, *block_num: int*) → int

Downloads a DB data into the AG asynchronously. A whole block (including header and footer) must be available into the user buffer.

Parameters

- **block_num** – New Block number (or -1)
- **data** – the user buffer

as_read_area(*area: str*, *dbnumber: int*, *start: int*, *size: int*, *wordlen: int*, *pusrdata*) → int

This is the main function to read data from a PLC. With it you can read DB, Inputs, Outputs, Merkers, Timers and Counters.

Parameters

- **area** – chosen memory_area
- **dbnumber** – The DB number, only used when area= S7AreaDB
- **start** – offset to start writing
- **size** – number of units to read
- **pusrdata** –
- **wordlen** –

as_write_area(*area: str*, *dbnumber: int*, *start: int*, *size: int*, *wordlen: int*, *pusrdata*) → int
This is the main function to write data into a PLC. It's the complementary function of Cli_ReadArea(), the parameters and their meanings are the same. The only difference is that the data is transferred from the buffer pointed by pUsrData into PLC.

Parameters

- **area** – chosen memory_area
- **dbnumber** – The DB number, only used when area= S7AreaDB
- **start** – offset to start writing

check_as_completion(*p_value*) → int
Method to check Status of an async request. Result contains if the check was successful, not the data value itself :param p_value: Pointer where result of this check shall be written. :return: 0 - Job is done successfully :return: 1 - Job is either pending or contains s7errors

create()
create a SNAP7 client.

db_get(*db_number: int*) → bytearray
Uploads a DB from AG.

db_read(*db_number: int*, *start: int*, *size: int*) → bytearray
This is a lean function of Cli_ReadArea() to read PLC DB.

Returns user buffer.

delete(*block_type: str*, *block_num: int*) → int
Deletes a block

Parameters

- **block_type** – Type of block
- **block_num** – Bloc number

destroy() → Optional[int]
destroy a client.

full_upload(*_type: str*, *block_num: int*) → Tuple[bytearray, int]
Uploads a full block body from AG. The whole block (including header and footer) is copied into the user buffer.

Parameters **block_num** – Number of Block

get_block_info(*blocktype: str*, *db_number: int*) → snap7.types.TS7BlockInfo
Returns the block information for the specified block.

get_connected() → bool
Returns the connection status

Returns a boolean that indicates if connected.

get_cpu_info() → snap7.types.S7CpuInfo
Retrieves CPU info from client

get_cpu_state() → str
Retrieves CPU state from client

get_param(*number: int*) → int
Reads an internal Client object parameter.

get_pdu_length() → int
Returns info about the PDU length.

get_plc_datetime() → datetime.datetime
Get date and time from PLC.

Returns date and time as datetime

list_blocks() → snap7.types.BlocksList
Returns the AG blocks amount divided by type.

Returns a snap7.types.BlocksList object.

list_blocks_of_type(*blocktype*, *size: int*) → Union[int, _ctypes.Array]
This function returns the AG list of a specified block type.

plc_cold_start() → int
cold starts a client

plc_hot_start() → int
hot starts a client

plc_stop() → int
stops a client

read_area(*area: str*, *dbnumber: int*, *start: int*, *size: int*) → bytearray
This is the main function to read data from a PLC. With it you can read DB, Inputs, Outputs, Merkers, Timers and Counters.

Parameters

- **dbnumber** – The DB number, only used when area= S7AreaDB
- **start** – offset to start writing
- **size** – number of units to read

read_multi_vars(*items*) → Tuple[int, snap7.types.S7DataItem]
This function read multiple variables from the PLC.

Parameters **items** – list of S7DataItem objects

Returns a tuple with the return code and a list of data items

set_connection_params(*address: str*, *local_tsap: int*, *remote_tsap: int*)
Sets internally (IP, LocalTSAP, RemoteTSAP) Coordinates. This function must be called just before Cli_Connect().

Parameters

- **address** – PLC/Equipment IPV4 Address, for example "192.168.1.12"
- **local_tsap** – Local TSAP (PC TSAP)
- **remote_tsap** – Remote TSAP (PLC TSAP)

set_connection_type(*connection_type: int*)
Sets the connection resource type, i.e the way in which the Clients connects to a PLC.

Parameters **connection_type** – 1 for PG, 2 for OP, 3 to 10 for S7 Basic

upload(*block_num: int*) → bytearray
Uploads a block body from AG

Parameters **block_num** – bytearray

wait_as_completion(*timeout: int*) → int
Snap7 Cli_WaitAsCompletion representative. :param timeout: ms to wait for async job :return: Result of request in int format

snap7.client.**error_wrap**(*func*)
Parses a s7 error code returned the decorated function.

CHAPTER 5

Server

Snap7 server used for mimicking a siemens 7 server.

class snap7.server.**Server**(*log: bool = True*)

A fake S7 server.

create()

create the server.

destroy()

destroy the server.

event_text(*event: snap7.types.SrvEvent*) → str

Returns a textual explanation of a given event object

Parameters **event** – an PSrvEvent struct object

Returns the error string

get_mask(*kind: int*) → ctypes.c_uint

Reads the specified filter mask.

get_param(*number*) → int

Reads an internal Server object parameter.

get_status() → Tuple[str, str, int]

Reads the server status, the Virtual CPU status and the number of the clients connected.

Returns server status, cpu status, client count

pick_event() → Optional[snap7.types.SrvEvent]

Extracts an event (if available) from the Events queue.

snap7.server.**error_wrap**(*func*)

Parses a s7 error code returned the decorated function.

CHAPTER 6

Partner

Snap7 code for partnering with a siemens 7 server.

This allows you to create a S7 peer to peer communication. Unlike the client-server model, where the client makes a request and the server replies to it, the peer to peer model sees two components with same rights, each of them can send data asynchronously. The only difference between them is the one who is requesting the connection.

class snap7.partner.**Partner**(*active: bool = False*)

A snap7 partner.

as_b_send() → int

Sends a data packet to the partner. This function is asynchronous, i.e. it terminates immediately, a completion method is needed to know when the transfer is complete.

b_recv() → int

Receives a data packet from the partner. This function is synchronous, it waits until a packet is received or the timeout supplied expires.

b_send() → int

Sends a data packet to the partner. This function is synchronous, i.e. it terminates when the transfer job (send+ack) is complete.

check_as_b_recv_completion() → int

Checks if a packed received was received.

check_as_b_send_completion() → Tuple[str, ctypes.c_int]

Checks if the current asynchronous send job was completed and terminates immediately.

create(*active: bool = False*)

Creates a Partner and returns its handle, which is the reference that you have to use every time you refer to that Partner.

Parameters **active** – 0

Returns a pointer to the partner object

destroy()

Destroy a Partner of given handle. Before destruction the Partner is stopped, all clients disconnected and all shared memory blocks released.

get_last_error() → ctypes.c_int
Returns the last job result.

get_param(*number*) → int
Reads an internal Partner object parameter.

get_stats() → Tuple[ctypes.c_uint, ctypes.c_uint, ctypes.c_uint, ctypes.c_uint]
Returns some statistics.

Returns a tuple containing bytes send, received, send errors, recv errors

get_status() → ctypes.c_int
Returns the Partner status.

get_times() → Tuple[ctypes.c_int, ctypes.c_int]
Returns the last send and recv jobs execution time in milliseconds.

set_recv_callback() → int
Sets the user callback that the Partner object has to call when a data packet is incoming.

set_send_callback() → int
Sets the user callback that the Partner object has to call when the asynchronous data sent is complete.

stop() → int
Stops the Partner, disconnects gracefully the remote partner.

snap7.partner.**error_wrap**(*func*)
Parses a s7 error code returned the decorated function.

CHAPTER 7

Logo

class snap7.logo.**Logo**

A snap7 Siemens Logo client: There are two main comfort functions available *Logo.read()* and *Logo.write()*. This functions realize a high level access to the VM addresses of the Siemens Logo just use the form:

- V10.3 for bit values
- V10 for the complete byte
- VW12 for a word (used for analog values)

For more information see examples for Siemens Logo 7 and 8

connect(*ip_address: str*, *tsap_snap7: int*, *tsap_logo: int*, *tcpport: int = 102*) → int

Connect to a Siemens LOGO server. Howto setup Logo communication configuration see: http://snap7.sourceforge.net/logo.html

Parameters

- **ip_address** – IP ip_address of server
- **tsap_snap7** – TSAP SNAP7 Client (e.g. 10.00 = 0x1000)
- **tsap_logo** – TSAP Logo Server (e.g. 20.00 = 0x2000)

create()

create a SNAP7 client.

db_read(*db_number: int*, *start: int*, *size: int*) → bytearray

This is a lean function of Cli_ReadArea() to read PLC DB.

Parameters

- **db_number** – for Logo only DB=1
- **start** – start address for Logo7 0..951 / Logo8 0..1469
- **size** – in bytes

Returns array of bytes

db_write(*db_number: int*, *start: int*, *data: bytearray*) → int
Writes to a DB object.

Parameters

- **db_number** – for Logo only DB=1
- **start** – start address for Logo7 0..951 / Logo8 0..1469
- **data** – bytearray

destroy()
destroy a client.

disconnect() → int
disconnect a client.

get_connected() → bool
Returns the connection status

Returns a boolean that indicates if connected.

get_param(*number*) → int
Reads an internal Logo object parameter.

Parameters **number** – Parameter type number

Returns Parameter value

read(*vm_address: str*)
Reads from VM addresses of Siemens Logo. Examples: read("V40") / read("VW64") / read("V10.2")

Parameters **vm_address** – of Logo memory (e.g. V30.1, VW32, V24)

Returns integer

set_connection_params(*ip_address: str*, *tsap_snap7: int*, *tsap_logo: int*)
Sets internally (IP, LocalTSAP, RemoteTSAP) Coordinates. This function must be called just before Cli_Connect().

Parameters

- **ip_address** – IP ip_address of server
- **tsap_snap7** – TSAP SNAP7 Client (e.g. 10.00 = 0x1000)
- **tsap_logo** – TSAP Logo Server (e.g. 20.00 = 0x2000)

set_connection_type(*connection_type: int*)
Sets the connection resource type, i.e the way in which the Clients connects to a PLC.

Parameters **connection_type** – 1 for PG, 2 for OP, 3 to 10 for S7 Basic

set_param(*number: int*, *value*)
Sets an internal Server object parameter.

Parameters

- **number** – Parameter type number
- **value** – Parameter value

write(*vm_address: str*, *value: int*) → int
Writes to VM addresses of Siemens Logo. Example: write("VW10", 200) or write("V10.3", 1)

Parameters

- **vm_address** – write offset

- **value** – integer

CHAPTER 8

Util

This module contains utility functions for working with PLC DB objects. There are functions to work with the raw bytearray data snap7 functions return In order to work with this data you need to make python able to work with the PLC bytearray data.

For example code see test_util.py and example.py in the example folder.

example:

```
spec/DB layout

# Byte index    Variable name  Datatype
layout="""
4              ID             INT
6              NAME           STRING[6]

12.0           testbool1      BOOL
12.1           testbool2      BOOL
12.2           testbool3      BOOL
12.3           testbool4      BOOL
12.4           testbool5      BOOL
12.5           testbool6      BOOL
12.6           testbool7      BOOL
12.7           testbool8      BOOL
13             testReal       REAL
17             testDword      DWORD
"""

client = snap7.client.Client()
client.connect('192.168.200.24', 0, 3)

# this looks confusing but this means uploading from the PLC to YOU
# so downloading in the PC world :)

all_data = client.upload(db_number)
```

(continues on next page)

(continued from previous page)

```
simple:

db1 = snap7.util.DB(
    db_number,              # the db we use
    all_data,               # bytearray from the plc
    layout,                 # layout specification DB variable data
                            # A DB specification is the specification of a
                            # DB object in the PLC you can find it using
                            # the dataview option on a DB object in PCS7

    17+2,                   # size of the specification 17 is start
                            # of last value
                            # which is a DWORD which is 2 bytes,

    1,                      # number of row's / specifications

    id_field='ID',          # field we can use to identify a row.
                            # default index is used
    layout_offset=4,        # sometimes specification does not start a 0
                            # like in our example
    db_offset=0             # At which point in 'all_data' should we start
                            # reading. if could be that the specification
                            # does not start at 0
)

Now we can use db1 in python as a dict. if 'ID' contains
the 'test' we can identify the 'test' row in the all_data bytearray

To test of you layout matches the data from the plc you can
just print db1[0] or db['test'] in the example

db1['test']['testbool1'] = 0

If we do not specify a id_field this should work to read out the
same data.

db1[0]['testbool1']

to read and write a single Row from the plc. takes like 5ms!

db1['test'].write()

db1['test'].read()
```

class snap7.util.**DB**(*db_number*, *bytearray_*, *specification*, *row_size*, *size*, *id_field=None*, *db_offset=0*, *layout_offset=0*, *row_offset=0*)

Manage a DB bytearray block given a specification of the Layout.

It is possible to have many repetitive instances of a specification this is called a "row".

probably most usecases there is just one row

db1[0]['testbool1'] = test db1.write() # puts data in plc

class snap7.util.**DB_Row**(*bytearray_*, *_specification*, *row_size=0*, *db_offset=0*, *layout_offset=0*, *row_offset=0*)

Provide ROW API for DB bytearray

export()

export dictionary with values

get_bytearray() → bytearray
return bytearray from self or DB parent

get_offset(*byte_index: Union[str, int]*) → int
Calculate correct beginning position for a row the db_offset = row_size * index

read(*client*)
read current data of db row from plc

write(*client*)
Write current data to db in plc

snap7.util.**get_bool**(*bytearray_: bytearray*, *byte_index: int*, *bool_index: int*) → bool
Get the boolean value from location in bytearray

snap7.util.**get_dint**(*bytearray_: bytearray*, *byte_index: int*)
Get dint value from bytearray. DINT (Double integer) 32bit 4 bytes Decimal number signed L#-2147483648 to L#2147483647

snap7.util.**get_int**(*bytearray_: bytearray*, *byte_index: int*)
Get int value from bytearray.

int are represented in two bytes

snap7.util.**get_real**(*bytearray_: bytearray*, *byte_index: int*)
Get real value. create float from 4 bytes

snap7.util.**get_sint**(*bytearray_: bytearray*, *byte_index: int*) → int
get the small int

Args:

bytearray_ (bytearray)

byte_index (int): index of the bytearray

Returns: int: small int (-127 - 128)

snap7.util.**get_string**(*bytearray_: bytearray*, *byte_index: int*, *max_size: int*) → str
parse string from bytearray

snap7.util.**get_usint**(*bytearray_: bytearray*, *byte_index: int*) → int
get the unsigned small int from the bytearray

Args:

bytearray_ (bytearray)

byte_index (int): index of the bytearray

Returns: int: unsigned small int (0 - 255)

snap7.util.**get_word**(*bytearray_: bytearray*, *byte_index: int*)
Get word value from bytearray. WORD 16bit 2bytes Decimal number unsigned B#(0,0) to B#(255,255) => 0 to 65535

snap7.util.**parse_specification**(*db_specification: str*) → collections.OrderedDict
Create a db specification derived from a dataview of a db in which the byte layout is specified

snap7.util.**set_bool**(*bytearray_: bytearray*, *byte_index: int*, *bool_index: int*, *value: bool*)
Set boolean value on location in bytearray

snap7.util.**set_dint**(*bytearray_: bytearray*, *byte_index: int*, *dint: int*)
Set value in bytearray to dint

snap7.util.**set_int**(*bytearray_: bytearray*, *byte_index: int*, *_int: int*)
Set value in bytearray to int

snap7.util.**set_real**(*bytearray_: bytearray*, *byte_index: int*, *real*)
Set Real value

make 4 byte data from real

snap7.util.**set_sint**(*bytearray_: bytearray*, *byte_index: int*, *_int*) → bytearray
set small int

Args:

bytearray_ (bytearray)

byte_index (int): index of the bytearray _int (int): small int (-128 - 127)

Returns: bytearray

snap7.util.**set_string**(*bytearray_: bytearray*, *byte_index: int*, *value: str*, *max_size: int*)
Set string value

Params value string data

Params max_size max possible string size

snap7.util.**set_usint**(*bytearray_: bytearray*, *byte_index: int*, *_int: int*) → bytearray
set unsigned small int

Args:

bytearray_ (bytearray): bytearray

byte_index (int): index of the bytearray _int (int): positive value to set (0 - 255)

Returns: bytearray: bytearray of the db

snap7.util.**set_word**(*bytearray_: bytearray*, *byte_index: int*, *_int: int*)
Set value in bytearray to word

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index

L

P

R

S

U

W