# python-snap7 Documentation

***Release 0.0rc0***

**Gijs Molenaar, Stephan Preeker**

**Jul 07, 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 (10 64 bit), OSX 10.15 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

# Binary Wheel Installation

We advice you to install python-snap7 using a binary wheel. The binary wheels should work on Windows 64x, OS X (intel) and Linux x64. python-snap7 is available on PyPI. You can install it by using pip:

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

CHAPTER 3

# Manual Installation (not recommended)

If you are running an unsupported platform you need to do a bit more work. This involves two steps. First, install the snap7 library, followed by the installation of the python-snap7 package.

## 3.1 Snap7

### 3.1.1 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
```

### 3.1.2 Windows

Download the zip file from the sourceforce page. Unzip the zip file, and copy `release\\Windows\\Win64\\snap7.dll` somewhere in your system PATH, for example `%systemroot%\System32\`. Alternatively you can copy the file somewhere on your file system and adjust the system PATH.

### 3.1.3 OSX

The snap7 library is available on Homebrew:

```
$ brew install snap7
```

### 3.1.4 Compile from source

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
```

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

## 3.2 Python-Snap7

Once snap7 is available in your library or system path, you can install it from the git repository or from a source tarball:

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

CHAPTER 4

# development

## 4.1 Github

We develop python-snap7 on github. If you have questions about python-snap7 please raise a question in the Q&A discussion sessions. If you have a bug or feature request for 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.

## 4.2 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.

## 4.3 Credits

python-snap7 is created by:

- Gijs Molenaar (gijs at pythonic dot nl)
- Stephan Preeker (stephan at preeker dot net)

Special thanks to:

- Davide Nardella for creating snap7
- Thomas Hergenhahn for his libnodave
- Thomas W for his S7comm wireshark plugin
- Fabian Beitler and Nikteliy for their contributions towards the 1.0 release
- Lautaro Nahuel Dapino for his contributions.

# CHAPTER 5

# Client

Snap7 client used for connection to a siemens 7 server.

**class** snap7.client.**Client**(*lib_location: Optional[str] = None*)

A snap7 client

### Examples

```
>>> import snap7
>>> client = snap7.client.Client()
>>> client.connect("127.0.0.1", 0, 0, 1012)
>>> client.get_connected()
True
>>> data = client.db_read(1, 0, 4)
>>> data
bytearray(b"\x00\x00\x00\x00")
>>> data[3] = 0b00000001
>>> data
bytearray(b'\x00\x00\x00\x01')
>>> data.db_write(1, 0, data)
```

**__init__**(*lib_location: Optional[str] = None*)

Creates a new *Client* instance.

**Parameters** **lib_location** – Full path to the snap7.dll file. Optional.

### Examples

```
>>> import snap7
>>> client = snap7.client.Client()  # If the `snap7.dll` file is in the path␣
↪location
>>> client = snap7.client.Client(lib_location="/path/to/snap7.dll")  # If the␣
↪`snap7.dll` file is in another location
```

(continues on next page)

(continued from previous page)

```
>>> client
<snap7.client.Client object at 0x0000028B257128E0>
```

**ab_read**(*start: int*, *size: int*) → bytearray
Reads a part of IPU area from a PLC.

**Parameters**

- **start** – byte index from where start to read.
- **size** – amount of bytes to read.

**Returns** Buffer with the data read.

**ab_write**(*start: int*, *data: bytearray*) → int
Writes a part of IPU area into a PLC.

**Parameters**

- **start** – byte index from where start to write.
- **data** – buffer with the data to be written.

**Returns** Snap7 code.

**as_ab_read**(*start: int*, *size: int*, *data*) → int
Reads a part of IPU area from a PLC asynchronously.

**Parameters**

- **start** – byte index from where start to read.
- **size** – amount of bytes to read.
- **data** – buffer where the data will be place.

**Returns** Snap7 code.

**as_ab_write**(*start: int*, *data: bytearray*) → int
Writes a part of IPU area into a PLC asynchronously.

**Parameters**

- **start** – byte index from where start to write.
- **data** – buffer with the data to be written.

**Returns** Snap7 code.

**as_compress**(*time: int*) → int
Performs the Compress action asynchronously.

**Parameters** **time** – timeout.

**Returns** Snap7 code.

**as_copy_ram_to_rom**(*timeout: int = 1*) → int
Performs the Copy Ram to Rom action asynchronously.

**Parameters** **timeout** – time to wait unly fail.

**Returns** Snap7 code.

**as_ct_read**(*start: int*, *amount: int*, *data*) → int
Reads counters from a PLC asynchronously.

**Parameters**

- **start** – byte index to start to read from.
- **amount** – amount of bytes to read.
- **data** – buffer where the value read will be place.

**Returns** Snap7 code.

**as_ct_write**(*start: int*, *amount: int*, *data: bytearray*) → int
Write counters into a PLC.

**Parameters**

- **start** – byte index to start to write from.
- **amount** – amount of bytes to write.
- **data** – buffer to be write.

**Returns** Snap7 code.

**as_db_fill**(*db_number: int*, *filler*) → int
Fills a DB in AG with a given byte.

**Parameters**

- **db_number** – number of DB to fill.
- **filler** – buffer to fill with.

**Returns** Snap7 code.

**as_db_get**(*db_number: int*, *_buffer*, *size*) → bytearray
Uploads a DB from AG using DBRead.

---

**Note:** This method will not work in 1200/1500.

---

**Parameters**

- **db_number** – number of DB to get.
- **_buffer** – buffer where the data read will be place.
- **size** – amount of bytes to be read.

**Returns** Snap7 code.

**as_db_read**(*db_number: int*, *start: int*, *size: int*, *data*) → _ctypes.Array
Reads a part of a DB from a PLC.

**Parameters**

- **db_number** – number of DB to be read.
- **start** – byte index from where start to read from.
- **size** – amount of bytes to read.
- **data** – buffer where the data read will be place.

**Returns** Snap7 code.

### Examples

```
>>> import ctypes
>>> data = (ctypes.c_uint8 * size_to_read)()  # In this ctypes array data␣
↪will be stored.
>>> result = client.as_db_read(1, 0, size_to_read, data)
>>> result  # 0 = success
0
```

**as_db_write**(*db_number: int*, *start: int*, *size: int*, *data*) → int
Writes a part of a DB into a PLC.

**Parameters**

- **db_number** – number of DB to be write.
- **start** – byte index from where start to write to.
- **size** – amount of bytes to write.
- **data** – buffer to be write.

**Returns** Snap7 code.

**as_download**(*data: bytearray*, *block_num: int*) → int
Download a block into AG asynchronously.

**Note:** A whole block (including header and footer) must be available into the user buffer.

**Parameters**

- **block_num** – new block number.
- **data** – buffer where the data will be place.

**Returns** Snap7 code.

**as_eb_read**(*start: int*, *size: int*, *data*) → int
Reads a part of IPI area from a PLC asynchronously.

**Parameters**

- **start** – byte index from where to start reading from.
- **size** – amount of bytes to read.
- **data** – buffer where the data read will be place.

**Returns** Snap7 code.

**as_eb_write**(*start: int*, *size: int*, *data: bytearray*) → int
Writes a part of IPI area into a PLC.

**Parameters**

- **start** – byte index from where to start writing from.
- **size** – amount of bytes to write.
- **data** – buffer to write.

**Returns** Snap7 code.

**as_full_upload**(*_type: str*, *block_num: int*) → int
Uploads a block from AG with Header and Footer infos.

---

**Note:** Upload means from PLC to PC.

---

**Parameters**

- **_type** – type of block.
- **block_num** – number of block to upload.

**Returns** Snap7 code.

**as_list_blocks_of_type**(*blocktype: str*, *data*, *count*) → int
Returns the AG blocks list of a given type.

**Parameters**

- **blocktype** – block type.
- **data** – buffer where the data will be place.
- **count** – pass.

**Returns** Snap7 code.

**Raises** `Snap7Exception` – if the *blocktype* is invalid

**as_mb_read**(*start: int*, *size: int*, *data*) → int
Reads a part of Merkers area from a PLC.

**Parameters**

- **start** – byte index from where to start to read from.
- **size** – amount of byte to read.
- **data** – buffer where the data read will be place.

**Returns** Snap7 code.

**as_mb_write**(*start: int*, *size: int*, *data: bytearray*) → int
Writes a part of Merkers area into a PLC.

**Parameters**

- **start** – byte index from where to start to write to.
- **size** – amount of byte to write.
- **data** – buffer to write.

**Returns** Snap7 code.

**as_read_area**(*area: snap7.types.Areas*, *dbnumber: int*, *start: int*, *size: int*, *wordlen: snap7.types.WordLen*, *pusrdata*) → int
Reads a data area from a PLC asynchronously. With it you can read DB, Inputs, Outputs, Merkers, Timers and Counters.

**Parameters**

- **area** – memory area to be read from.
- **dbnumber** – The DB number, only used when area=Areas.DB
- **start** – offset to start writing

- **size** – number of units to read
- **pusrdata** – buffer where the data will be place.
- **wordlen** – length of the word to be read.

**Returns** Snap7 code.

**as_read_szl**(*ssl_id: int*, *index: int*, *s7_szl: snap7.types.S7SZL*, *size*) → int
Reads a partial list of given ID and Index.

**Parameters**

- **ssl_id** – TODO
- **index** – TODO
- **s7_szl** – TODO
- **size** – TODO

**Returns** Snap7 code.

**as_read_szl_list**(*szl_list*, *items_count*) → int
Reads the list of partial lists available in the CPU.

**Parameters**

- **szl_list** – TODO
- **items_count** – TODO

**Returns** Snap7 code.

**as_tm_read**(*start: int*, *amount: int*, *data*) → bytearray
Reads timers from a PLC.

**Parameters**

- **start** – byte index to start read from.
- **amount** – amount of bytes to read.
- **data** – buffer where the data will be placed.

**Returns** Snap7 code.

**as_tm_write**(*start: int*, *amount: int*, *data: bytearray*) → int
Write timers into a PLC.

**Parameters**

- **start** – byte index to start writing to.
- **amount** – amount of bytes to write.
- **data** – buffer to write.

**Returns** Snap7 code.

**as_upload**(*block_num: int*, *_buffer*, *size*) → int
Uploads a block from AG.

---

**Note:** Uploads means from PLC to PC.

---

**Parameters**

- **block_num** – block number to upload.
- **_buffer** – buffer where the data will be place.
- **size** – amount of bytes to uplaod.

**Returns** Snap7 code.

**as_write_area**(*area: snap7.types.Areas*, *dbnumber: int*, *start: int*, *size: int*, *wordlen: snap7.types.WordLen*, *pusrdata*) → int

Writes a data area into a PLC asynchronously.

**Parameters**

- **area** – memory area to be written.
- **dbnumber** – The DB number, only used when area=Areas.DB
- **start** – offset to start writing.
- **size** – amount of bytes to be written.
- **wordlen** – length of the word to be written.
- **pusrdata** – buffer to be written.

**Returns** Snap7 code.

**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

**Parameters** **p_value** – Pointer where result of this check shall be written.

**Returns** Snap7 code. If 0 - Job is done successfully. If 1 - Job is either pending or contains s7errors

**copy_ram_to_rom**(*timeout: int = 1*) → int

Performs the Copy Ram to Rom action.

**Parameters** **timeout** – timeout time.

**Returns** Snap7 code.

**create**()

Creates a SNAP7 client.

**ct_read**(*start: int*, *amount: int*) → bytearray

Reads counters from a PLC.

**Parameters**

- **start** – byte index to start read from.
- **amount** – amount of bytes to read.

**Returns** Buffer read.

**ct_write**(*start: int*, *amount: int*, *data: bytearray*) → int

Write counters into a PLC.

**Parameters**

- **start** – byte index to start write to.
- **amount** – amount of bytes to write.
- **data** – buffer data to write.

**Returns** Snap7 code.

**db_fill** (*db_number: int*, *filler: int*) → int
Fills a DB in AG with a given byte.

**Parameters**

- **db_number** – db number to fill.
- **filler** – value filler.

**Returns** Snap7 code.

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

---

**Note:** This method can't be use for 1200/1500 PLCs.

---

**Parameters** **db_number** – db number to be read from.

**Returns** Buffer with the data read.

### Example

```
>>> import snap7
>>> client = snap7.client.Client()
>>> client.connect("192.168.0.1", 0, 0)
>>> buffer = client.db_get(1)  # reads the db number 1.
>>> buffer
bytearray(b"\x00\x00\x00\x00\x00\x00\x00\x00...<truncated>\x00\x00")
```

**db_read** (*db_number: int*, *start: int*, *size: int*) → bytearray
Reads a part of a DB from a PLC

---

**Note:** Use it only for reading DBs, not Marks, Inputs, Outputs.

---

**Parameters**

- **db_number** – number of the DB to be read.
- **start** – byte index from where is start to read from.
- **size** – amount of bytes to be read.

**Returns** Buffer read.

### Example

```
>>> import snap7
>>> client = snap7.client.Client()
>>> client.connect("192.168.0.1", 0, 0)
>>> buffer = client.db_read(1, 10, 4)  # reads the db number 1 starting from␣
↪the byte 10 until byte 14.
>>> buffer
bytearray(b'\x00\x00')
```

**delete**(*block_type: str*, *block_num: int*) → int
Delete a block into AG.

**Parameters**

- **block_type** – type of block.
- **block_num** – block number.

**Returns** Error code from snap7 library.

**destroy**() → Optional[int]
Destroys the Client object.

**Returns** Error code from snap7 library.

### Examples

```
>>> client.destroy()
640719840
```

**eb_read**(*start: int*, *size: int*) → bytearray
Reads a part of IPI area from a PLC.

**Parameters**

- **start** – byte index to start read from.
- **size** – amount of bytes to read.

**Returns** Data read.

**eb_write**(*start: int*, *size: int*, *data: bytearray*) → int
Writes a part of IPI area into a PLC.

**Parameters**

- **start** – byte index to be written.
- **size** – amount of bytes to write.
- **data** – data to write.

**Returns** Snap7 code.

**error_text**(*error: int*) → str
Returns a textual explanation of a given error number.

**Parameters** **error** – error number.

**Returns** Text error.

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

**Parameters**

- **_type** – type of block.
- **block_num** – number of block.

**Returns** Tuple of the buffer and size.

**get_block_info**(*blocktype: str*, *db_number: int*) → snap7.types.TS7BlockInfo
Returns detailed information about a block present in AG.

**Parameters**

- **blocktype** – specified block type.
- **db_number** – number of db to get information from.

**Returns** Structure of information from block.

**Raises** `Snap7Exception` – if the *blocktype* is not valid.

### Examples

```
>>> block_info = client.get_block_info("DB", 1)
>>> print(block_info)
Block type: 10
Block number: 1
Block language: 5
Block flags: 1
MC7Size: 100
Load memory size: 192
Local data: 0
SBB Length: 20
Checksum: 0
Version: 1
Code date: b'1999/11/17'
Interface date: b'1999/11/17'
Author: b''
Family: b''
Header: b''
```

**get_connected**() → bool
Returns the connection status

**Note:** Sometimes returns True, while connection is lost.

**Returns** True if is connected, otherwise false.

**get_cp_info**() → snap7.types.S7CpInfo
Returns some information about the CP (communication processor).

**Returns** Structure object containing the CP information.

**get_cpu_info**() → snap7.types.S7CpuInfo
Returns some information about the AG.

**Returns** data structure with the information.

**Return type** `S7CpuInfo`

### Examples

```
>>> cpu_info = client.get_cpu_info()
>>> print(cpu_info)
"<S7CpuInfo ModuleTypeName: b'CPU 315-2 PN/DP'
    SerialNumber: b'S C-C2UR28922012'
```

(continues on next page)

(continued from previous page)

```
    ASName: b'SNAP7-SERVER' Copyright: b'Original Siemens Equipment'
    ModuleName: b'CPU 315-2 PN/DP'>
```

**get_cpu_state**() → str
Returns the CPU status (running/stopped)

**Returns** Description of the cpu state.

**Raises** `Snap7Exception` – if the cpu state is invalid.

### Examples

```
>>> client.get_cpu_statE()
'S7CpuStatusRun'
```

**get_exec_time**() → int
Returns the last job execution time in milliseconds.

**Returns** Execution time value.

**get_last_error**() → int
Returns the last job result.

**Returns** Returns the last error value.

**get_order_code**() → snap7.types.S7OrderCode
Returns the CPU order code.

**Returns** Order of the code in a structure object.

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

**Parameters** **number** – number of argument to be read.

**Returns** Value of the param read.

**get_pdu_length**() → int
Returns info about the PDU length (requested and negotiated).

**Returns** PDU length.

### Examples

```
>>> client.get_pdu_length()
480
```

**get_pg_block_info**(*block: bytearray*) → snap7.types.TS7BlockInfo
Returns detailed information about a block loaded in memory.

**Parameters** **block** – buffer where the data will be place.

**Returns** Structure object that contains the block information.

**get_plc_datetime**() → datetime.datetime
Returns the PLC date/time.

**Returns** Date and time as datetime

### Examples

```
>>> client.get_plc_datetime()
datetime.datetime(2021, 4, 6, 12, 12, 36)
```

**get_protection**() → snap7.types.S7Protection
Gets the CPU protection level info.

**Returns** Structure object with protection attributes.

**iso_exchange_buffer**(*data: bytearray*) → bytearray
Exchanges a given S7 PDU (protocol data unit) with the CPU.

**Parameters** **data** – buffer to exchange.

**Returns** Snap7 code.

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

**Returns** Block list structure object.

### Examples

```
>>> block_list = client.list_blocks()
>>> print(block_list)
<block list count OB: 0 FB: 0 FC: 0 SFB: 0 SFC: 0x0 DB: 1 SDB: 0>
```

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

**Parameters**

- **blocktype** – specified block type.
- **size** – size of the block type.

**Returns** If size is 0, it returns a 0, otherwise an *Array* of specified block type.

**Raises** `Snap7Exception` – if the *blocktype* is not valid.

**mb_read**(*start: int*, *size: int*) → bytearray
Reads a part of Merkers area from a PLC.

**Parameters**

- **start** – byte index to be read from.
- **size** – amount of bytes to read.

**Returns** Buffer with the data read.

**mb_write**(*start: int*, *size: int*, *data: bytearray*) → int
Writes a part of Merkers area into a PLC.

**Parameters**

- **start** – byte index to be written.
- **size** – amount of bytes to write.
- **data** – buffer to write.

**Returns** Snap7 code.

**plc_cold_start**() → int
Puts the CPU in RUN mode performing a COLD START.

**Returns** Error code from snap7 library.

**plc_hot_start**() → int
Puts the CPU in RUN mode performing an HOT START.

**Returns** Error code from snap7 library.

**plc_stop**() → int
Puts the CPU in STOP mode

**Returns** Error code from snap7 library.

**read_area**(*area: snap7.types.Areas*, *dbnumber: int*, *start: int*, *size: int*) → bytearray
Reads a data area from a PLC With it you can read DB, Inputs, Outputs, Merkers, Timers and Counters.

**Parameters**

- **area** – area to be read from.
- **dbnumber** – number of the db to be read from. In case of Inputs, Marks or Outputs, this should be equal to 0.
- **start** – byte index to start reading.
- **size** – number of bytes to read.

**Returns** Buffer with the data read.

**Raises** `ValueError` – if the area is not defined in the *Areas*

### Example

```
>>> import snap7
>>> client = snap7.client.Client()
>>> client.connect("192.168.0.1", 0, 0)
>>> buffer = client.read_area(snap7.types.Areas.DB, 1, 10, 4)  # Reads the DB␣
↪number 1 from the byte 10 to the byte 14.
>>> buffer
bytearray(b'\x00\x00')
```

**read_multi_vars**(*items*) → Tuple[int, snap7.types.S7DataItem]
Reads different kind of variables from a PLC simultaneously.

**Parameters** **items** – list of items to be read.

**Returns** Tuple with the return code from the snap7 library and the list of items.

**read_szl**(*ssl_id: int*, *index: int = 0*) → snap7.types.S7SZL
Reads a partial list of given ID and Index.

**Parameters**

- **ssl_id** – ssl id to be read.
- **index** – index to be read.

**Returns** SZL structure object.

**read_szl_list**() → bytearray
Reads the list of partial lists available in the CPU.

**Returns** Buffer read.

**set_connection_params**(*address: str*, *local_tsap: int*, *remote_tsap: int*) → None
Sets internally (IP, LocalTSAP, RemoteTSAP) Coordinates.

---

**Note:** 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)

**Raises**

- `ValueError` – if the *address* is not a valid IPV4.
- `Snap7Exception` – if the result of setting the connection params is different than 0.

**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

**Raises** `Snap7Exception` – if the result of setting the connection type is different than 0.

**set_plc_system_datetime**() → int
Sets the PLC date/time with the host (PC) date/time.

**Returns** Snap7 code.

**tm_read**(*start: int*, *amount: int*) → bytearray
Reads timers from a PLC.

**Parameters**

- **start** – byte index from where is start to read from.
- **amount** – amount of byte to be read.

**Returns** Buffer read.

**tm_write**(*start: int*, *amount: int*, *data: bytearray*) → int
Write timers into a PLC.

**Parameters**

- **start** – byte index from where is start to write to.
- **amount** – amount of byte to be written.
- **data** – data to be write.

**Returns** Snap7 code.

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

---

**Note:** Upload means from the PLC to the PC.

---

**Parameters** **block_num** – block to be upload.

**Returns** Buffer with the uploaded block.

**wait_as_completion**(*timeout: int*) → int
Snap7 Cli_WaitAsCompletion representative.

**Parameters** **timeout** – ms to wait for async job

**Returns** Snap7 code.

**write_multi_vars**(*items: List[snap7.types.S7DataItem]*) → int
Writes different kind of variables into a PLC simultaneously.

**Parameters** **items** – list of items to be written.

**Returns** Snap7 code.

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

CHAPTER 6

# Server

Snap7 server used for mimicking a siemens 7 server.

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

A fake S7 server.

**__init__**(*log: bool = True*)

**Create a fake S7 server. set log to false if you want to disable** event logging to python logging.

**Parameters** **log** – *True* for enabling the event logging. Optinoal.

**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.

**Parameters** **kind** –

**Returns** Mask

**get_param**(*number*) → int

Reads an internal Server object parameter.

**Parameters** **number** – number of the parameter to be set.

**Returns** Value of the 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.

**Returns** Server event.

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

snap7.server.**mainloop**(*tcpport: int = 1102*, *init_standard_values: bool = False*)
Init a fake Snap7 server with some default values.

**Parameters**

- **tcpport** – port that the server will listen.
- **init_standard_values** – if *True* will init some defaults values to be read on DB0.

# CHAPTER 7

# 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 8

# 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:

### Notes

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

**__init__**()

Creates a new instance of *Logo*

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

Connect to a Siemens LOGO server.

### Notes

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)

**Returns** Error code from snap7 library.

**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

**Returns** Error code from snap7 library.

**destroy**() → int
Destroy a client.

**Returns** Error code from snap7 library.

**disconnect**() → int
Disconnect a client.

**Returns** Error code from snap7 library.

**get_connected**() → bool
Returns the connection status

### Notes

**This function has a bug, that returns *True* when the connection** is lost. This comes from the original *snap7 library*.

**Returns** True 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.

### Notes

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)

**Raises**

- `ValueError` – if the *ip_address* is not an IPV4.
- `Snap7Exception` – if the snap7 error code is diferent from 0.

**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

**Raises** `Snap7Exception` – if the snap7 error code is diferent from 0.

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

**Parameters**

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

**Returns** Error code from snap7 library.

**write**(*vm_address: str*, *value: int*) → int
Writes to VM addresses of Siemens Logo.

**Parameters**

- **vm_address** – write offset
- **value** – integer

### Examples

```
>>> write("VW10", 200) or write("V10.3", 1)
```

CHAPTER 9

# 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(client)
```

**class** snap7.util.**DB**(*db_number: int*, *bytearray_: bytearray*, *specification: str*, *row_size: int*, *size: int*, *id_field: Optional[str] = None*, *db_offset: Optional[int] = 0*, *layout_offset: Optional[int] = 0*, *row_offset: Optional[int] = 0*, *area: Optional[snap7.types.Areas] = <Areas.DB: 132>*)

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

**bytearray_**
buffer data from the PLC.

**specification**

layout of the DB Rows.

**row_size**
bytes size of a db row.

**layout_offset**
at which byte in the row specificaion we start reading the data.

**db_offset**
at which byte in the db starts reading.

### Examples

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

**__init__**(*db_number: int*, *bytearray_: bytearray*, *specification: str*, *row_size: int*, *size: int*, *id_field: Optional[str] = None*, *db_offset: Optional[int] = 0*, *layout_offset: Optional[int] = 0*, *row_offset: Optional[int] = 0*, *area: Optional[snap7.types.Areas] = <Areas.DB: 132>*)
Creates a new instance of the *Row* class.

**Parameters**

- **db_number** – number of the DB to read from. This value should be 0 if area!=Areas.DB.
- **bytearray** – initial buffer read from the PLC.
- **specification** – layout of the PLC memory.
- **row_size** – bytes size of a db row.
- **size** – lenght of the memory area.
- **id_field** – name to reference the row. Optional.
- **db_offset** – at which byte in the db starts reading.
- **layout_offset** – at which byte in the row specificaion we start reading the data.
- **row_offset** – offset between rows.
- **area** – which memory area this row is representing.

**make_rows**()
Make each row for the DB.

**set_data**(*bytearray_: bytearray*)
Set the new buffer data from the PLC to the current instance.

**Parameters** **bytearray** – buffer to save.

**Raises** `TypeError` – if *bytearray_* is not an instance of `bytearray`

**class** snap7.util.**DB_Row**(*bytearray_: bytearray*, *_specification: str*, *row_size: Optional[int] = 0*, *db_offset: int = 0*, *layout_offset: int = 0*, *row_offset: Optional[int] = 0*, *area: Optional[snap7.types.Areas] = <Areas.DB: 132>*)
Provide ROW API for DB bytearray

**bytearray_**
reference to the data of the parent DB.

**_specification**
row specification layout.

**__getitem__**(*key*)
Get a specific db field

**__init__**(*bytearray_: bytearray*, *_specification: str*, *row_size: Optional[int] = 0*, *db_offset: int = 0*, *layout_offset: int = 0*, *row_offset: Optional[int] = 0*, *area: Optional[snap7.types.Areas] = <Areas.DB: 132>*)
Creates a new instance of the *DB_Row* class.

**Parameters**

- **bytearray** – reference to the data of the parent DB.
- **_specification** – row specification layout.
- **row_size** – Amount of bytes of the row.
- **db_offset** – at which byte in the db starts reading.
- **layout_offset** – at which byte in the row specificaion we start reading the data.
- **row_offset** – offset between rows.
- **area** – which memory area this row is representing.

**Raises** `TypeError` – if *bytearray_* is not an instance of `bytearray` or *DB*.

**export**() → Dict[str, Union[str, int, float, bool, datetime.datetime]]
Export dictionary with values

**Returns** dictionary containing the values of each value of the row.

**get_bytearray**() → bytearray
Gets bytearray from self or DB parent

**Returns** Buffer data corresponding to the row.

**get_offset**(*byte_index: Union[str, int]*) → int

**Calculate correct beginning position for a row** the db_offset = row_size * index

**Parameters** **byte_index** – byte index from where to start reading from.

**Returns** Amount of bytes to ignore.

**get_value**(*byte_index: Union[str, int]*, *type_: str*) → Union[ValueError, int, float, str, datetime.datetime]
Gets the value for a specific type.

**Parameters**

- **byte_index** – byte index from where start reading.
- **type** – type of data to read.

**Raises**

- `Snap7Exception` – if reading a *string* when checking the lenght of the string.
- `ValueError` – if the *type_* is not handled.

**Returns** Value read according to the *type_*

**read**(*client: snap7.client.Client*) → None
Read current data of db row from plc.

**Parameters** **client** – `Client` snap7 instance.

**Raises**

- `TypeError` – if the *_bytearray* is not an instance of *DB* class.
- `ValueError` – if the *row_size* is less than 0.

**set_value**(*byte_index: Union[str, int], type: str, value: Union[bool, str, int, float]*) → Optional[bytearray]

Sets the value for a specific type in the specified byte index.

**Parameters**

- **byte_index** – byte index to start writing to.
- **type** – type of value to write.
- **value** – value to write.

**Raises**

- `Snap7Exception` – if reading a *string* when checking the lenght of the string.
- `ValueError` – if the *type_* is not handled.

**Returns** Buffer data with the value written. Optional.

**unchanged**(*bytearray_: bytearray*) → bool

Checks if the bytearray is the same

**Parameters** **bytearray** – buffer of data to check.

**Returns** True if the current *bytearray_* is equal to the new one. Otherwise is False.

**write**(*client: snap7.client.Client*) → None

Write current data to db in plc

**Parameters** **client** – `Client` snap7 instance.

**Raises**

- `TypeError` – if the *_bytearray* is not an instance of *DB* class.
- `ValueError` – if the *row_size* is less than 0.

snap7.util.**get_bool**(*bytearray_: bytearray*, *byte_index: int*, *bool_index: int*) → bool

Get the boolean value from location in bytearray

**Parameters**

- **bytearray** – buffer data.
- **byte_index** – byte index to read from.
- **bool_index** – bit index to read from.

**Returns** True if the bit is 1, else 0.

### Examples

```
>>> buffer = bytearray([0b00000001])  # Only one byte length
>>> get_bool(buffer, 0, 0)  # The bit 0 starts at the right.
    True
```

snap7.util.**get_byte**(*bytearray_: bytearray*, *byte_index: int*) → int

Get byte value from bytearray.

### Notes

WORD 8bit 1bytes Decimal number unsigned B#(0) to B#(255) => 0 to 255

**Parameters**

- **bytearray** – buffer to be read from.
- **byte_index** – byte index to be read.

**Returns** value get from the byte index.

snap7.util.**get_dint**(*bytearray_: bytearray*, *byte_index: int*) → int
Get dint value from bytearray.

### Notes

Datatype *dint* consists in 4 bytes in the PLC. Maximum possible value is 2147483647. Lower posible value is -2147483648.

**Parameters**

- **bytearray** – buffer to read.
- **byte_index** – byte index from where to start reading.

**Returns** Value read.

### Examples

```
>>> import struct
>>> data = bytearray(4)
>>> data[:] = struct.pack(">i", 2147483647)
>>> snap7.util.get_dint(data, 0)
    2147483647
```

snap7.util.**get_dword**(*bytearray_: bytearray*, *byte_index: int*) → int
Gets the dword from the buffer.

### Notes

Datatype *dword* consists in 8 bytes in the PLC. The maximum value posible is *4294967295*

**Parameters**

- **bytearray** – buffer to read.
- **byte_index** – byte index from where to start reading.

**Returns** Value read.

### Examples

```
>>> data = bytearray(8)
>>> data[:] = b"\x12\x34\xAB\xCD"
>>> snap7.util.get_dword(data, 0)
    4294967295
```

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

### Notes

Datatype *int* in the PLC is represented in two bytes

**Parameters**

- **bytearray** – buffer to read from.
- **byte_index** – byte index to start reading from.

**Returns** Value read.

### Examples

```
>>> data = bytearray([0, 255])
>>> snap7.util.get_int(data, 0)
    255
```

snap7.util.**get_real**(*bytearray_: bytearray*, *byte_index: int*) → float
Get real value.

### Notes

Datatype *real* is represented in 4 bytes in the PLC. The packed representation uses the *IEEE 754 binary32*.

**Parameters**

- **bytearray** – buffer to read from.
- **byte_index** – byte index to reading from.

**Returns** Real value.

### Examples

```
>>> data = bytearray(b'B\xf6\xa4Z')
>>> snap7.util.get_real(data, 0)
    123.32099914550781
```

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

### Notes

Datatype *sint* (Small int) consists in 1 byte in the PLC. Maximum value posible is 127. Lowest value posible is -128.

**Parameters**

- **bytearray** – buffer to read from.
- **byte_index** – byte index from where to start reading.

**Returns** Value read.

### Examples

```
>>> data = bytearray([127])
>>> snap7.util.get_sint(data, 0)
    127
```

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

### Notes

The first byte of the buffer will contain the max size posible for a string. The second byte contains the length of the string that contains.

**Parameters**

- **bytearray** – buffer from where to get the string.
- **byte_index** – byte index from where to start reading.
- **max_size** – maximum possible string size.

**Returns** String value.

### Examples

```
>>> data = bytearray([254, len("hello world")] + [ord(letter) for letter in
↪"hello world"])
>>> snap7.util.get_string(data, 0, 255)
'hello world'
```

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

### Notes

Datatype *usint* (Unsigned small int) consists on 1 byte in the PLC. Maximum posible value is 255. Lower posible value is 0.

**Parameters**

- **bytearray** – buffer to read from.
- **byte_index** – byte index from where to start reading.

**Returns** Value read.

### Examples

```
>>> data = bytearray([255])
>>> snap7.util.get_usint(data, 0)
    255
```

snap7.util.**get_word**(*bytearray_: bytearray*, *byte_index: int*) → int
Get word value from bytearray.

### Notes

WORD 16bit 2bytes Decimal number unsigned B#(0,0) to B#(255,255) => 0 to 65535

**Parameters**

- **bytearray** – buffer to get the word from.
- **byte_index** – byte index from where start reading from.

**Returns** Word value.

### Examples

```
>>> data = bytearray([0, 100])  # two bytes for a word
>>> snap7.util.get_word(data, 0)
    100
```

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

**Parameters** **db_specification** – string formatted table with the indexes, aliases and types.

**Returns** Parsed DB specification.

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

**Parameters**

- **bytearray** – buffer to write to.
- **byte_index** – byte index to write to.
- **bool_index** – bit index to write to.
- **value** – value to write.

### Examples

```
>>> buffer = bytearray([0b00000000])
>>> set_bool(buffer, 0, 0, True)
>>> buffer
    bytearray(b"\x01")
```

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

**Parameters**

- **bytearray** – buffer to write to.
- **byte_index** – byte index to write.
- **_int** – value to write.

**Returns** buffer with the written value.

### Examples

```
>>> buffer = bytearray([0b00000000])
>>> set_byte(buffer, 0, 255)
    bytearray(b"\xFF")
```

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

### Notes

Datatype *dint* consists in 4 bytes in the PLC. Maximum possible value is 2147483647. Lower posible value is -2147483648.

**Parameters**

- **bytearray** – buffer to write.
- **byte_index** – byte index from where to start writing.

### Examples

```
>>> data = bytearray(4)
>>> snap7.util.set_dint(data, 0, 2147483647)
>>> data
    bytearray(b'\x7f\xff\xff\xff')
```

snap7.util.**set_dword**(*bytearray_: bytearray*, *byte_index: int*, *dword: int*)
Set a DWORD to the buffer.

### Notes

Datatype *dword* consists in 8 bytes in the PLC. The maximum value posible is *4294967295*

**Parameters**

- **bytearray** – buffer to write to.
- **byte_index** – byte index from where to writing reading.
- **dword** – value to write.

### Examples

```
>>> data = bytearray(4)
>>> snap7.util.set_dword(data,0, 4294967295)
>>> data
    bytearray(b'\xff\xff\xff\xff')
```

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

### Notes

An datatype *int* in the PLC consists of two *bytes*.

**Parameters**

- **bytearray** – buffer to write on.
- **byte_index** – byte index to start writing from.
- **_int** – int value to write.

**Returns** Buffer with the written value.

### Examples

```
>>> data = bytearray(2)
>>> snap7.util.set_int(data, 0, 255)
    bytearray(b'\x00\xff')
```

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

### Notes

Datatype *real* is represented in 4 bytes in the PLC. The packed representation uses the *IEEE 754 binary32*.

**Parameters**

- **bytearray** – buffer to write to.
- **byte_index** – byte index to start writing from.
- **real** – value to be written.

**Returns** Buffer with the value written.

### Examples

```
>>> data = bytearray(4)
>>> snap7.util.set_real(data, 0, 123.321)
    bytearray(b'B\xf6\xa4Z')
```

snap7.util.**set_sint**(*bytearray_: bytearray*, *byte_index: int*, *_int*) → bytearray
Set small int to the buffer.

### Notes

Datatype *sint* (Small int) consists in 1 byte in the PLC. Maximum value posible is 127. Lowest value posible is -128.

**Parameters** **bytearray** – buffer to write to.

byte_index: byte index from where to start writing. _int: value to write.

**Returns** Buffer with the written value.

### Examples

```
>>> data = bytearray(1)
>>> snap7.util.set_sint(data, 0, 127)
    bytearray(b'\x7f')
```

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

**Parameters**

- **bytearray** – buffer to write to.
- **byte_index** – byte index to start writing from.
- **value** – string to write.
- **max_size** – maximum possible string size.

**Raises**

- TypeError – if the *value* is not a str.
- ValueError – if the length of the *value* is larger than the *max_size*.

### Examples

```
>>> data = bytearray(20)
>>> snap7.util.set_string(data, 0, "hello world", 255)
>>> data
    bytearray(b'\x00\x0bhello world\x00\x00\x00\x00\x00\x00\x00')
```

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

### Notes

Datatype *usint* (Unsigned small int) consists on 1 byte in the PLC. Maximum posible value is 255. Lower posible value is 0.

**Parameters** **bytearray** – buffer to write.

byte_index: byte index from where to start writing. _int: value to write.

**Returns** Buffer with the written value.

### Examples

```
>>> data = bytearray(1)
>>> snap7.util.set_usint(data, 0, 255)
    bytearray(b'\xff')
```

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

### Notes

Word datatype is 2 bytes long.

**Parameters**

- **bytearray** – buffer to be written.
- **byte_index** – byte index to start write from.
- **_int** – value to be write.

**Returns** buffer with the written value

snap7.util.**utc2local**(*utc: Union[datetime.date, datetime.datetime]*) → Union[datetime.datetime, datetime.date]

Returns the local datetime

**Parameters** **utc** – UTC type date or datetime.

**Returns** Local datetime.

CHAPTER 10

# Indices and tables

- genindex
- modindex
- search

# Python Module Index

## S

# Index

## Symbols



## A



## B



## C



## D

## T



## U



## W