# steem-python Documentation

***Release 0.18.9***

**furion@steemit.com**

**Aug 03, 2017**

# Contents

`steem-python` is the official STEEM library for Python. It comes with a BIP38 encrypted wallet and a practical CLI utility called *stemepy*.

The Steem library has been designed to allow developers to easily access its routines and make use of the network without dealing with all the releated blockchain technology and cryptography. This library can be used to do anything that is allowed according to the Steem blockchain protocol.

CHAPTER 1

# Installation

To install the library, simply run:

```
pip install -U steem
```

CHAPTER 2

# Getting Started

## Installation

*steem-python* requires Python 3.5 or higher. We don't recommend usage of Python that ships with OS. If you're just looking for a quick and easy cross-platform solution, feel free to install Python 3.x via easy to use Anaconda installer.

Afterwards, you can install *steem-python* with *pip*:

```
$ pip install steem
```

You can also perform the installation manually using *setup.py*:

```
$ git clone https://github.com/steemit/steem-python
$ cd steem-python
$ make install
```

### Upgrade

```
$ pip install -U steem
```

#### Namespace Collision

If you have used a piston stack before v0.5, please remove it before installing official version of `python-steem`.

```
curl https://raw.githubusercontent.com/steemit/steem-python/master/scripts/nuke_
↪legacy.sh | sh
```

# steempy CLI

*steempy* is a convenient CLI utility that enables you to manage your wallet, transfer funds, check balances and more.

## Using the Wallet

*steempy* lets you leverage your BIP38 encrypted wallet to perform various actions on your accounts.

The first time you use *steempy*, you will be prompted to enter a password. This password will be used to encrypt the *steempy* wallet, which contains your private keys.

You can change the password via *changewalletpassphrase* command.

```
steempy changewalletpassphrase
```

From this point on, every time an action requires your private keys, you will be prompted ot enter this password (from CLI as well as while using *steem* library).

To bypass password entry, you can set an environmnet variable `UNLOCK`.

```
UNLOCK=mysecretpassword steempy transfer 100 STEEM <recipient>
```

## Common Commands

First, you may like to import your Steem account:

```
steempy importaccount
```

You can also import individual private keys:

```
steempy addkey <private_key>
```

Listing accounts:

```
steempy listaccounts
```

Show balances:

```
steempy balance account_name1 account_name2
```

Sending funds:

```
steempy transfer --account <account_name> 100 STEEM <recipient_name> memo
```

Upvoting a post:

```
steempy upvote --account <account_name> https://steemit.com/funny/@mynameisbrian/the-
↪content-stand-a-comic
```

## Setting Defaults

For a more convenient use of `steempy` as well as the `steem` library, you can set some defaults. This is especially useful if you have a single Steem account.

```
steempy set default_account furion
steempy set default_vote_weight 100

steempy config
 +---------------------+---------+
 | Key                 | Value   |
 +---------------------+---------+
 | default_account     | furion  |
 | default_vote_weight | 100     |
 +---------------------+---------+
```

If you've set up your *default_account*, you can now send funds by omitting this field:

```
steempy transfer 100 STEEM <recipient_name> memo
```

## Help

You can see all available commands with `steempy -h`

```
~ % steempy -h
usage: steempy [-h] [--node NODE] [--no-broadcast] [--no-wallet] [--unsigned]
               [--expires EXPIRES] [--verbose VERBOSE] [--version]
               {set,config,info,changewalletpassphrase,addkey,delkey,getkey,listkeys,
↪listaccounts,upvote,downvote,transfer,powerup,powerdown,powerdownroute,convert,
↪balance,interest,permissions,allow,disallow,newaccount,importaccount,updatememokey,
↪approvewitness,disapprovewitness,sign,broadcast,orderbook,buy,sell,cancel,resteem,
↪follow,unfollow,setprofile,delprofile,witnessupdate,witnesscreate}
               ...

Command line tool to interact with the Steem network

positional arguments:
  {set,config,info,changewalletpassphrase,addkey,delkey,getkey,listkeys,listaccounts,
↪upvote,downvote,transfer,powerup,powerdown,powerdownroute,convert,balance,interest,
↪permissions,allow,disallow,newaccount,importaccount,updatememokey,approvewitness,
↪disapprovewitness,sign,broadcast,orderbook,buy,sell,cancel,resteem,follow,unfollow,
↪setprofile,delprofile,witnessupdate,witnesscreate}
                        sub-command help
    set                 Set configuration
    config              Show local configuration
    info                Show basic STEEM blockchain info
    changewalletpassphrase
                        Change wallet password
    addkey              Add a new key to the wallet
    delkey              Delete keys from the wallet
    getkey              Dump the privatekey of a pubkey from the wallet
    listkeys            List available keys in your wallet
    listaccounts        List available accounts in your wallet
    upvote              Upvote a post
    downvote            Downvote a post
    transfer            Transfer STEEM
    powerup             Power up (vest STEEM as STEEM POWER)
    powerdown           Power down (start withdrawing STEEM from steem POWER)
    powerdownroute      Setup a powerdown route
    convert             Convert STEEMDollars to Steem (takes a week to settle)
    balance             Show the balance of one more more accounts
```

```
    interest              Get information about interest payment
    permissions           Show permissions of an account
    allow                 Allow an account/key to interact with your account
    disallow              Remove allowance an account/key to interact with your
                          account
    newaccount            Create a new account
    importaccount         Import an account using a passphrase
    updatememokey         Update an account's memo key
    approvewitness        Approve a witnesses
    disapprovewitness     Disapprove a witnesses
    sign                  Sign a provided transaction with available and
                          required keys
    broadcast             broadcast a signed transaction
    orderbook             Obtain orderbook of the internal market
    buy                   Buy STEEM or SBD from the internal market
    sell                  Sell STEEM or SBD from the internal market
    cancel                Cancel order in the internal market
    resteem               Resteem an existing post
    follow                Follow another account
    unfollow              unfollow another account
    setprofile            Set a variable in an account's profile
    delprofile            Set a variable in an account's profile
    witnessupdate         Change witness properties
    witnesscreate         Create a witness

optional arguments:
  -h, --help              show this help message and exit
  --node NODE             URL for public Steem API (default:
                          "https://steemd.steemit.com")
  --no-broadcast, -d      Do not broadcast anything
  --no-wallet, -p         Do not load the wallet
  --unsigned, -x          Do not try to sign the transaction
  --expires EXPIRES, -e EXPIRES
                          Expiration time in seconds (defaults to 30)
  --verbose VERBOSE, -v VERBOSE
                          Verbosity
  --version               show program's version number and exit
```

# Examples

## Syncing Blockchain to a Flat File

Here is a relatively simple script built on top of `steem-python` that will let you sync STEEM blockchain into a simple file. You can run this script as many times as you like, and it will continue from the last block it synced.

```
import json
import os
from contextlib import suppress
from steem.blockchain import Blockchain


def get_last_line(filename):
    if os.path.isfile(filename):
        with open(filename, 'rb') as f:
            f.seek(-2, 2)
```

```
            while f.read(1) != b"\n":
                f.seek(-2, 1)
            return f.readline()


def get_previous_block_num(block):
    if not block:
        return -1

    if type(block) == bytes:
        block = block.decode('utf-8')

    if type(block) == str:
        block = json.loads(block)

    return int(block['previous'][:8], base=16)


def run(filename):
    b = Blockchain()
    # automatically resume from where we left off
    # previous + last + 1
    start_block = get_previous_block_num(get_last_line(filename)) + 2
    with open(filename, 'a+') as file:
        for block in b.stream_from(start_block=start_block, full_
↪blocks=True):
            file.write(json.dumps(block, sort_keys=True) + '\n')


if __name__ == '__main__':
    output_file = '/home/user/Downloads/steem.blockchain.json'
    with suppress(KeyboardInterrupt):
        run(output_file)
```

To see how many blocks we currently have, we can simply perform a line count.

```
wc -l steem.blockchain.json
```

We can also inspect an arbitrary block, and pretty-print it. *Replace 10000 with desired block_number + 1.*

```
sed '10000q;d' steem.blockchain.json | python -m json.tool
```

## Witness Killswitch

Occasionally things go wrong: software crashes, servers go down... One of the main roles for STEEM witnesses is to reliably mint blocks. This script acts as a kill-switch to protect the network from missed blocks and prevents embarrassment when things go totally wrong.

```
import time
from steem import Steem

steem = Steem()

# variables
disable_after = 10  # disable witness after 10 blocks are missed
witness_name = 'furion'
```

```
witness_url = "https://steemit.com/steemit/@furion/power-down-no-more"
witness_props = {
    "account_creation_fee": "0.500 STEEM",
    "maximum_block_size": 65536,
    "sbd_interest_rate": 15,
}


def total_missed():
    return steem.get_witness_by_account(witness_name)['total_missed']


if __name__ == '__main__':
    treshold = total_missed() + disable_after
    while True:
        if total_missed() > treshold:
            tx = steem.commit.witness_update(
                signing_key=None,
                url=witness_url,
                props=witness_props,
                account=witness_name)

            print("Witness %s Disabled!" % witness_name)
            quit(0)

        time.sleep(60)
```

## Batching Operations

Most of the time each transaction contains only one operation (for example, an upvote, a transfer or a new post). We can however cram multiple operations in a single transaction, to achieve better efficiency and size reduction.

This script will also teach us how to create and sign transactions ourselves.

```
from steem.transactionbuilder import TransactionBuilder
from steembase import operations

# lets create 3 transfers, to 3 different people
transfers = [
    {
        'from': 'richguy',
        'to': 'recipient1',
        'amount': '0.001 STEEM',
        'memo': 'Test Transfer 1'
    },
    {
        'from': 'richguy',
        'to': 'recipient2',
        'amount': '0.002 STEEM',
        'memo': 'Test Transfer 2'
    },
    {
        'from': 'richguy',
        'to': 'recipient3',
        'amount': '0.003 STEEM',
        'memo': 'Test Transfer 3'
    }
```

```
]

# now we can construct the transaction
# we will set no_broadcast to True because
# we don't want to really send funds, just testing.
tb = TransactionBuilder(no_broadcast=True)

# lets serialize our transfers into a format Steem can understand
operations = [operations.Transfer(**x) for x in transfers]

# tell TransactionBuilder to use our serialized transfers
tb.appendOps(operations)

# we need to tell TransactionBuilder about
# everyone who needs to sign the transaction.
# since all payments are made from `richguy`,
# we just need to do this once
tb.appendSigner('richguy', 'active')

# sign the transaction
tb.sign()

# broadcast the transaction (publish to steem)
# since we specified no_broadcast=True earlier
# this method won't actually do anything
tx = tb.broadcast()
```

## Simple Voting Bot

Here is a simple bot that will reciprocate by upvoting all new posts that mention us. Make sure to set `whoami` to your Steem username before running.

```
from contextlib import suppress

from steem.blockchain import Blockchain
from steem.post import Post


def run():
    # upvote posts with 30% weight
    upvote_pct = 30
    whoami = 'my-steem-username'

    # stream comments as they are published on the blockchain
    # turn them into convenient Post objects while we're at it
    b = Blockchain()
    stream = map(Post, b.stream(filter_by=['comment']))

    for post in stream:
        if post.json_metadata:
            mentions = post.json_metadata.get('users', [])

            # if post mentions more than 10 people its likely spam
            if mentions and len(mentions) < 10:
                post.upvote(weight=upvote_pct, voter=whoami)
```

```
if __name__ == '__main__':
    with suppress(KeyboardInterrupt):
        run()
```

CHAPTER 3

# Digging Deeper

## Steem - Your Starting Point

### Quick Start

You can start using the library with just a few lines of code, as seen in this quick example:

```
# first, we initialize Steem class
from steem import Steem
s = Steem()
```

```
# check @ned's balance
>>> s.get_account('ned')['sbd_balance']
'980.211 SBD'

# lets send $1.0 SBD to @ned
>>> s.commit.transfer(to='ned', amount=1, asset='SBD', account='furion')
{'expiration': '2017-03-12T17:54:43',
 'extensions': [],
 'operations': [['transfer',
   {'amount': '1.000 SBD', 'from': 'furion', 'memo': '', 'to': 'ned'}]],
 'ref_block_num': 23008,
 'ref_block_prefix': 961695589,
 'signatures': [
↪'1f1322be9ca0c22b27c0385c929c9863901ac78cdaedea2162024ea040e22c4f8b542c02d96cbc761cbe4a188a932bc715
↪']}

# yup, its there
>>> s.get_account('ned')['sbd_balance']
'981.211 SBD'
```

### Importing your Steem Account

*steem-python* comes with a BIP38 encrypted wallet, which holds your private keys.

Alternatively, you can also pass required WIF's to `Steem()` initializer.

```
from steem import Steem
s = Steem(keys=['<private_posting_key>', '<private_active_key>'])
```

Using the encrypted wallet is however a recommended way.

Please check *steempy CLI* to learn how to set up the wallet.

### Interfacing with steemd

`Steem()` inherits API methods from `Steemd`, which can be called like so:

```
s = Steem()

s.get_account('ned')
s.get_block(8888888)
s.get_content('author', 'permlink')
s.broadcast_transaction(...)
# and many more
```

You can see the list of available methods by calling `help(Steem)`. If a method is not available trough the Python API, we can call it manually using `s.exec()`:

```
s = Steem()

# this call
s.get_followers('furion', 'abit', 'blog', 10)

# is same as
s.exec('get_followers',
       'furion', 'abit', 'blog', 10,
        api='follow_api')
```

### Commit and Wallet

`Steem()` comes equipped with `Commit` and `Wallet`, accessible via dot-notation.

```
s = Steem()

# accessing Commit methods
s.commit.transfer(...)

# accessing Wallet methods
s.wallet.get_active_key_for_account(...)
```

Please check *Transactions and Accounts* documentation to learn more.

## Steem

As displayed in the *Quick Start* above, `Steem` is the main class of this library. It acts as a gateway to other components, such as `Steemd`, `Commit`, `Wallet` and `HttpClient`.

Any arguments passed to `Steem` as `kwargs` will naturally flow to sub-components. For example, if we initialize Steem with `steem = Steem(no_broadcast=True)`, the `Commit` instance is configured to not broadcast any transactions. This is very useful for testing.

**class** `steem.steem.`**`Steem`**(*nodes=None*, *no_broadcast=False*, ***kwargs*)

Connect to the Steem network.

**Parameters**

- **`nodes`** (*`list`*) – A list of Steem HTTP RPC nodes to connect to. If not provided, official Steemit nodes will be used.
- **`debug`** (*`bool`*) – Elevate logging level to *logging.DEBUG*. Defaults to *logging.INFO*.
- **`no_broadcast`** (*`bool`*) – If set to `True`, committal actions like sending funds will have no effect (simulation only).

Optional Arguments (kwargs):

**Parameters**

- **`keys`** (*`list`*) – A list of wif keys. If provided, the Wallet will use these keys rather than the ones found in BIP38 encrypted wallet.
- **`unsigned`** (*`bool`*) – (Defaults to False) Use this for offline signing.
- **`expiration`** (*`int`*) – (Defualts to 60) Size of window in seconds that the transaction needs to be broadcasted in, before it expires.

**Returns** Steemd class instance. It can be used to execute commands against steem node.

### Example

If you would like to override the official Steemit nodes (default), you can pass your own. When currently used node goes offline, `Steemd` will automatically fail-over to the next available node.

```
nodes = [
    'https://steemd.yournode1.com',
    'https://steemd.yournode2.com',
]

s = Steemd(nodes)
```

## Steemd API

Steemd contains API generating utilities. `Steemd`'s methods will be automatically available to `Steem()` classes. See *Steem - Your Starting Point*.

**class** `steem.steemd.`**`Steemd`**(*nodes=None*, ***kwargs*)

Connect to the Steem network.

**Parameters** **`nodes`** (*`list`*) – A list of Steem HTTP RPC nodes to connect to. If not provided, official Steemit nodes will be used.

**Returns** Steemd class instance. It can be used to execute commands against steem node.

### Example

If you would like to override the official Steemit nodes (default), you can pass your own. When currently used node goes offline, `Steemd` will automatically fail-over to the next available node.

```
nodes = [
    'https://steemd.yournode1.com',
    'https://steemd.yournode2.com',
]

s = Steemd(nodes)
```

**broadcast_block**(*block: steem.block.Block*)

**broadcast_transaction**(*signed_transaction: steembase.transactions.SignedTransaction*)

**broadcast_transaction_synchronous**(*signed_transaction: steembase.transactions.SignedTransaction*)

**chain_params**
Identify the connected network. This call returns a dictionary with keys chain_id, prefix, and other chain specific settings

**get_account**(*account: str*)
Lookup account information such as user profile, public keys, balances, etc.

**Parameters** **account** (*str*) – STEEM username that we are looking up.

**Returns** Account information.

**Return type** dict

**get_account_bandwidth**(*account: str*, *bandwidth_type: object*)

**get_account_count**()
How many accounts are currently registered on STEEM?

**get_account_history**(*account: str*, *index_from: int*, *limit: int*)
History of all operations for a given account.

**Parameters**

- **account** (*str*) – STEEM username that we are looking up.
- **index_from** (*int*) – The highest database index we take as a starting point.
- **limit** (*int*) – How many items are we interested in.

**Returns** List of operations.

**Return type** list

### Example

To get the latest (newest) operations from a given user `furion`, we should set the `index_from` to -1. This is the same as saying *give me the highest index there is*.

```
s.get_account_history('furion', index_from=-1, limit=3)
```

This will yield 3 recent operations like so:

```
[[69974,
  {'block': 9941972,
   'op': ['vote',
    {'author': 'breezin',
     'permlink': 'raising-children-is-not-childsplay-pro-s-and-con-s-of-being-
↪a-young-parent',
     'voter': 'furion',
     'weight': 900}],
   'op_in_trx': 0,
   'timestamp': '2017-03-06T17:09:48',
   'trx_id': '87f9176faccc7096b5ffb5d12bfdb41b3c0b2955',
   'trx_in_block': 5,
   'virtual_op': 0}],
 [69975,
  {'block': 9942005,
   'op': ['curation_reward',
    {'comment_author': 'leongkhan',
     'comment_permlink': 'steem-investor-report-5-march-2017',
     'curator': 'furion',
     'reward': '112.397602 VESTS'}],
   'op_in_trx': 1,
   'timestamp': '2017-03-06T17:11:30',
   'trx_id': '0000000000000000000000000000000000000000',
   'trx_in_block': 5,
   'virtual_op': 0}],
 [69976,
  {'block': 9942006,
   'op': ['vote',
    {'author': 'ejhaasteem',
     'permlink': 'life-of-fishermen-in-aceh',
     'voter': 'furion',
     'weight': 100}],
   'op_in_trx': 0,
   'timestamp': '2017-03-06T17:11:30',
   'trx_id': '955018ac8efe298bd90b45a4fbd15b9df7e00be4',
   'trx_in_block': 7,
   'virtual_op': 0}]]
```

If we want to query for a particular range of indexes, we need to consider both *index_from* and *limit* fields. Remember, *index_from* works backwards, so if we set it to 100, we will get items *100, 99, 98, 97....*

For example, if we'd like to get the first 100 operations the user did, we would write:

```
s.get_account_history('furion', index_from=100, limit=100)
```

We can get the next 100 items by running:

```
s.get_account_history('furion', index_from=200, limit=100)
```

**get_account_references**(*account_id: int*)

**get_account_reputations**(*account: str*, *limit: int*)

**get_account_votes**(*account: str*)

All votes the given account ever made.

Returned votes are in the following format:

```
{'authorperm': 'alwaysfelicia/time-line-of-best-times-to-post-on-steemit-
↪mystery-explained',
'percent': 100,
'rshares': 709227399,
'time': '2016-08-07T16:06:24',
'weight': '3241351576115042'},
```

**Parameters** **account** (*str*) – STEEM username that we are looking up.

**Returns** List of votes.

**Return type** list

**get_accounts**(*account_names: list*)
Lookup account information such as user profile, public keys, balances, etc.

This method is same as `get_account`, but supports querying for multiple accounts at the time.

**get_active_votes**(*author: str*, *permlink: str*)
Get all votes for the given post.

**Parameters**

- **author** (*str*) – OP's STEEM username.
- **permlink** (*str*) – Post identifier following the username. It looks like slug-ified title.

**Returns** List of votes.

**Return type** list

### Example

```
s.get_active_votes('mynameisbrian', 'steemifying-idioms-there-s-no-use-crying-
↪over-spilt-milk')
```

Output:

```
[{'percent': 10000,
  'reputation': '36418980678',
  'rshares': 356981288,
  'time': '2017-03-06T20:04:18',
  'voter': 'dailystuff',
  'weight': '2287202760855'},
  ...
 {'percent': 10000,
  'reputation': 3386400109,
  'rshares': 364252169,
  'time': '2017-03-06T19:32:45',
  'voter': 'flourish',
  'weight': '2334690471157'}]
```

**get_active_witnesses**()
Get a list of currently active witnesses.

**get_all_usernames**(*last_user=''*)
Fetch the full list of STEEM usernames.

**get_api_by_name**(*api_name: str*)

**get_block**(*block_num: int*)
Get the full block, transactions and all, given a block number.

**Parameters** **block_num** (*int*) – Block number.

**Returns** Block in a JSON compatible format.

**Return type** dict

### Example

```
s.get_block(8888888)
```

```
{'extensions': [],
 'previous': '0087a2372163ff5c5838b09589ce281d5a564f66',
 'timestamp': '2017-01-29T02:47:33',
 'transaction_merkle_root': '4ddc419e531cccee6da660057d606d11aab9f3a5',
 'transactions': [{'expiration': '2017-01-29T02:47:42',
   'extensions': [],
   'operations': [['comment',
     {'author': 'hilarski',
      'body': 'https://media.giphy.com/media/RAx4Xwh1OPHji/giphy.gif',
      'json_metadata': '{"tags":["motocross"],"image":["https://media.giphy.
↪com/media/RAx4Xwh1OPHji/giphy.gif"],"app":"steemit/0.1"}',
      'parent_author': 'b0y2k',
      'parent_permlink': 'ama-supercross-round-4-phoenix-2017',
      'permlink': 're-b0y2k-ama-supercross-round-4-phoenix-2017-
↪20170129t024725575z',
      'title': ''}]],
   'ref_block_num': 41495,
   'ref_block_prefix': 2639073901,
   'signatures': [
↪'2058b69f4c15f704a67a7b5a7996a9c9bbfd39c639f9db19b99ecad8328c4ce3610643f8d1b6424c352df1206
↪']}],
 'witness': 'chainsquad.com',
 'witness_signature':
↪'1f115745e3f6fee95124164f4b57196c0eda2a700064faa97d0e037d3554ee2d5b618e6bfd457473783e8b833
↪'}
```

**get_block_header**(*block_num: int*)
Get block headers, given a block number.

**Parameters** **block_num** (*int*) – Block number.

**Returns** Block headers in a JSON compatible format.

**Return type** dict

### Example

```
s.get_block_headers(8888888)
```

```
{'extensions': [],
 'previous': '0087a2372163ff5c5838b09589ce281d5a564f66',
 'timestamp': '2017-01-29T02:47:33',
```

```
 'transaction_merkle_root': '4ddc419e531cccee6da660057d606d11aab9f3a5',
 'witness': 'chainsquad.com'}
```

**get_blocks**(*block_nums: typing.List[int]*)
Fetch multiple blocks from steemd at once, given a range.

**Parameters** **block_nums** (*list*) – A list of all block numbers we would like to tech.

**Returns** An ensured and ordered list of all *get_block* results.

**Return type** dict

**get_blocks_range**(*start: int*, *end: int*)
Fetch multiple blocks from steemd at once, given a range.

**Parameters**

- **start** (*int*) – The number of the block to start with
- **end** (*int*) – The number of the block at the end of the range. Not included in results.

**Returns** An ensured and ordered list of all *get_block* results.

**Return type** dict

**get_blog**(*account: str*, *entry_id: int*, *limit: int*)

**get_blog_authors**(*blog_account: str*)

**get_blog_entries**(*account: str*, *entry_id: int*, *limit: int*)

**get_chain_properties**()
Get witness elected chain properties.

```
{'account_creation_fee': '30.000 STEEM',
 'maximum_block_size': 65536,
 'sbd_interest_rate': 250}
```

**get_comment_discussions_by_payout**(*discussion_query: dict*)

**get_config**()
Get internal chain configuration.

**get_content**(*author: str*, *permlink: str*)

**get_content_replies**(*author: str*, *permlink: str*)

**get_conversion_requests**(*account: str*)

**get_current_median_history_price**()
Get the average STEEM/SBD price.

This price is based on moving average of witness reported price feeds.

```
{'base': '0.093 SBD', 'quote': '1.010 STEEM'}
```

**get_discussions_by_active**(*discussion_query: dict*)

**get_discussions_by_author_before_date**(*author: str*, *start_permlink: str*, *before_date: steembase.types.PointInTime*, *limit: int*)

**get_discussions_by_blog**(*discussion_query: dict*)

**get_discussions_by_cashout**(*discussion_query: dict*)

**get_discussions_by_children**(*discussion_query: dict*)

**get_discussions_by_comments**(*discussion_query: dict*)

**get_discussions_by_created**(*discussion_query: dict*)

**get_discussions_by_feed**(*discussion_query: dict*)

**get_discussions_by_hot**(*discussion_query: dict*)

**get_discussions_by_payout**(*discussion_query: dict*)

**get_discussions_by_promoted**(*discussion_query: dict*)

**get_discussions_by_trending**(*discussion_query: dict*)

**get_discussions_by_votes**(*discussion_query: dict*)

**get_dynamic_global_properties**()

**get_escrow**(*from_account: str*, *escrow_id: int*)

**get_expiring_vesting_delegations**(*account: str*, *start: steembase.types.PointInTime*, *limit: int*)

**get_feed**(*account: str*, *entry_id: int*, *limit: int*)

**get_feed_entries**(*account: str*, *entry_id: int*, *limit: int*)

**get_feed_history**()
Get the hourly averages of witness reported STEEM/SBD prices.

```
{'current_median_history': {'base': '0.093 SBD', 'quote': '1.010 STEEM'},
 'id': 0,
 'price_history': [{'base': '0.092 SBD', 'quote': '1.010 STEEM'},
  {'base': '0.093 SBD', 'quote': '1.020 STEEM'},
  {'base': '0.093 SBD', 'quote': '1.010 STEEM'},
  {'base': '0.094 SBD', 'quote': '1.020 STEEM'},
  {'base': '0.093 SBD', 'quote': '1.010 STEEM'},
```

**get_follow_count**(*account: str*)

**get_followers**(*account: str*, *start_follower: str*, *follow_type: str*, *limit: int*)

**get_following**(*account: str*, *start_follower: str*, *follow_type: str*, *limit: int*)

**get_hardfork_version**()
Get the current version of the chain.

---

**Note:** This is not the same as latest minor version.

---

**get_key_references**(*public_keys: typing.List[str]*)

**get_liquidity_queue**(*start_account: str*, *limit: int*)
Get the liquidity queue.

**Warning:** This feature is currently not in use, and might be deprecated in the future.

**get_market_history**(*bucket_seconds: int*, *start: steembase.types.PointInTime*, *end: steembase.types.PointInTime*)
Returns the market history for the internal SBD:STEEM market.

**get_market_history_buckets**()
Returns the bucket seconds being tracked by the plugin.

**get_next_scheduled_hardfork**()

**get_open_orders**(*account: str*)

**get_ops_in_block**(*block_num: int*, *virtual_only: bool*)

**get_order_book**(*limit: int*)
Get the internal market order book.

This method will return both bids and asks.

**Parameters** **limit** (*int*) – How many levels deep into the book to show.

**Returns** Order book.

**Return type** dict

### Example

```
s.get_order_book(2)
```

Outputs:

```
{'asks': [{'created': '2017-03-06T21:29:54',
   'order_price': {'base': '513.571 STEEM', 'quote': '50.000 SBD'},
   'real_price': '0.09735752213423265',
   'sbd': 50000,
   'steem': 513571},
  {'created': '2017-03-06T21:01:39',
   'order_price': {'base': '63.288 STEEM', 'quote': '6.204 SBD'},
   'real_price': '0.09802806219188472',
   'sbd': 6204,
   'steem': 63288}],
 'bids': [{'created': '2017-03-06T21:29:51',
   'order_price': {'base': '50.000 SBD', 'quote': '516.503 STEEM'},
   'real_price': '0.09680485882947436',
   'sbd': 50000,
   'steem': 516503},
  {'created': '2017-03-06T17:30:24',
   'order_price': {'base': '36.385 SBD', 'quote': '379.608 STEEM'},
   'real_price': '0.09584887568228277',
   'sbd': 36385,
   'steem': 379608}]}
```

**get_owner_history**(*account: str*)

**get_post_discussions_by_payout**(*discussion_query: dict*)

**get_posts**(*limit=10*, *sort='hot'*, *category=None*, *start=None*)
Get multiple posts in an array.

**Parameters**

- **limit** (*int*) – Limit the list of posts by `limit`
- **sort** (*str*) – Sort the list by "recent" or "payout"
- **category** (*str*) – Only show posts in this category
- **start** (*str*) – Show posts after this post. Takes an identifier of the form `@author/permlink`

**get_potential_signatures**(*signed_transaction: steembase.transactions.SignedTransaction*)

**get_promoted**()
Get promoted posts

**get_reblogged_by**(*author: str*, *permlink: str*)

**get_recent_trades**(*limit: int*) → typing.List[typing.Any]
Returns the N most recent trades for the internal SBD:STEEM market.

**get_recovery_request**(*account: str*)

**get_replies**(*author*, *skip_own=True*)
Get replies for an author

**Parameters**

- **author** (*str*) – Show replies for this author
- **skip_own** (*bool*) – Do not show my own replies

**get_replies_by_last_update**(*account: str*, *start_permlink: str*, *limit: int*)

**get_required_signatures**(*signed_transaction: steembase.transactions.SignedTransaction*, *available_keys: list*)

**get_reward_fund**(*fund_name: str = 'post'*)
Get details for a reward fund.

Right now the only pool available is 'post'.

### Example

```
s.get_reward_fund('post')
```

```
{'content_constant': '2000000000000',
 'id': 0,
 'last_update': '2017-04-09T19:18:57',
 'name': 'post',
 'percent_content_rewards': 10000,
 'percent_curation_rewards': 2500,
 'recent_claims': '1097112250115858684077192815608 4',
 'reward_balance': '555660.895 STEEM'}
```

**get_savings_withdraw_from**(*account: str*)

**get_savings_withdraw_to**(*account: str*)

**get_state**(*path: str*)

**get_tags_used_by_author**(*account: str*)

**get_ticker**()
Returns the market ticker for the internal SBD:STEEM market.

**get_trade_history**(*start: steembase.types.PointInTime*, *end: steembase.types.PointInTime*, *limit: int*)
Returns the trade history for the internal SBD:STEEM market.

**get_transaction**(*transaction_id: str*)

**get_transaction_hex**(*signed_transaction: steembase.transactions.SignedTransaction*)

**get_trending_tags**(*after_tag: str*, *limit: int*)

**get_version**()
Get steemd version of the node currently connected to.

**get_vesting_delegations**(*account: str*, *from_account: str*, *limit: int*)

**get_volume**()
Returns the market volume for the past 24 hours.

**get_withdraw_routes**(*account: str*, *withdraw_route_type: str*)

**get_witness_by_account**(*account: str*)

**get_witness_count**()

**get_witness_schedule**()

**get_witnesses**(*witness_ids: list*)

**get_witnesses_by_vote**(*from_account: str*, *limit: int*)

**head_block_number**
Newest block number.

**last_irreversible_block_num**
Newest irreversible block number.

**login**(*username: str*, *password: str*)

**lookup_account_names**(*account_names: list*)

**lookup_accounts**(*after: typing.Union[str, int]*, *limit: int*) → typing.List[str]
Get a list of usernames from all registered accounts.

**Parameters**

- **after** (*str, int*) – Username to start with. If '', 0 or -1, it will start at beginning.
- **limit** (*int*) – How many results to return.

**Returns** List of usernames in requested chunk.

**Return type** list

**lookup_witness_accounts**(*from_account: str*, *limit: int*)

**set_max_block_age**(*max_block_age: int*)

**stream_comments**(**args*, ***kwargs*)
Generator that yields posts when they come in

To be used in a for loop that returns an instance of *Post()*.

**verify_account_authority**(*account: str*, *keys: list*)

**verify_authority**(*signed_transaction: steembase.transactions.SignedTransaction*)

## Setting Custom Nodes

There are 3 ways in which you can set custom `steemd` nodes to use with `steem-python`.

**1. Global, permanent override:** You can use `steempy set nodes` command to set one or more node URLs. The nodes need to be separated with comma (,) and shall contain no whitespaces.

```
~ % steempy config
+---------------------+--------+
| Key                 | Value  |
+---------------------+--------+
| default_vote_weight | 100    |
| default_account     | furion |
+---------------------+--------+
~ % steempy set nodes https://gtg.steem.house:8090/
~ % steempy config
+---------------------+-------------------------------+
| Key                 | Value                         |
+---------------------+-------------------------------+
| default_account     | furion                        |
| default_vote_weight | 100                           |
| nodes               | https://gtg.steem.house:8090/ |
+---------------------+-------------------------------+
~ % steempy set nodes https://gtg.steem.house:8090/,https://steemd.steemit.
↪com
~ % steempy config
+---------------------+-------------------------------------------------------
↪----+
| Key                 | Value                                                 
↪    |
+---------------------+-------------------------------------------------------
↪----+
| nodes               | https://gtg.steem.house:8090/,https://steemd.steemit.
↪com |
| default_vote_weight | 100                                                   
↪    |
| default_account     | furion                                                
↪    |
+---------------------+-------------------------------------------------------
↪----+
~ %
```

To reset this config run `steempy set nodes ''`.

**2. For Current Python Process:** You can override default *Steemd* instance for current Python process, by overriding the *instance* singleton. You should execute the following code when your program starts, and from there on out, all classes (Blockchain, Account, Post, etc) will use this as their default instance.

```
from steem.steemd import Steemd
from steem.instance import set_shared_steemd_instance

steemd_nodes = [
    'https://gtg.steem.house:8090',
    'https://steemd.steemit.com',
]
set_shared_steemd_instance(Steemd(nodes=steemd_nodes))
```

**3. For Specific Class Instance:** Every class that depends on steemd comes with a `steemd_instance` argument. You can override said steemd instance, for any class you're initializing (and its children).

This is useful when you want to contain a modified `steemd` instance to an explicit piece of code (ie. for testing).

```
from steem.steemd import Steemd
from steem.account import Account
from steem.Blockchain import Blockchain
```

```
steemd_nodes = [
    'https://gtg.steem.house:8090',
    'https://steemd.steemit.com',
]
custom_instance = Steemd(nodes=steemd_nodes)

account = Account('furion', steemd_instance=custom_instance)
blockchain = Blockchain('head', steemd_instance=custom_instance)
```

# Transactions and Accounts

## Commit

The Commit class contains helper methods for *posting, voting, transferring funds, updating witnesses* and more. You don't have to use this class directly, all of its methods are accessible trough main `Steem` class.

```
# accessing commit methods trough Steem
s = Steem()
s.commit.transfer(...)

# is same as
c = Commit(steem=Steem())
c.transfer(..)
```

**class** steem.steem.**Commit**(*steemd_instance=None*, *no_broadcast=False*, ***kwargs*)

Commit things to the Steem network.

This class contains helper methods to construct, sign and broadcast common transactions, such as posting, voting, sending funds, etc.

**Parameters**

- **steemd** (Steemd) – Steemd node to connect to*
- **offline** (*bool*) – Do **not** broadcast transactions! *(optional)*
- **debug** (*bool*) – Enable Debugging *(optional)*
- **keys** (*list,dict,string*) – Predefine the wif keys to shortcut the wallet database

Three wallet operation modes are possible:

•**Wallet Database**: Here, the steemlibs load the keys from the locally stored wallet SQLite database (see `storage.py`). To use this mode, simply call `Steem()` without the `keys` parameter

•**Providing Keys**: Here, you can provide the keys for your accounts manually. All you need to do is add the wif keys for the accounts you want to use as a simple array using the `keys` parameter to `Steem()`.

•**Force keys**: This more is for advanced users and requires that you know what you are doing. Here, the `keys` parameter is a dictionary that overwrite the `active`, `owner`, `posting` or `memo` keys for any account. This mode is only used for *foreign* signatures!

**allow**(*foreign*, *weight=None*, *permission='posting'*, *account=None*, *threshold=None*)

Give additional access to an account by some other public key or account.

**Parameters**

- **foreign** (*str*) – The foreign account that will obtain access

- **weight** (*int*) – (optional) The weight to use. If not define, the threshold will be used. If the weight is smaller than the threshold, additional signatures will be required. (defaults to threshold)
- **permission** (*str*) – (optional) The actual permission to modify (defaults to `posting`)
- **account** (*str*) – (optional) the account to allow access to (defaults to `default_account`)
- **threshold** (*int*) – The threshold that needs to be reached by signatures to be able to interact

**approve_witness** (*witness*, *account=None*, *approve=True*)

Vote **for** a witness. This method adds a witness to your set of approved witnesses. To remove witnesses see `disapprove_witness`.

**Parameters**

- **witness** (*str*) – witness to approve
- **account** (*str*) – (optional) the account to allow access to (defaults to `default_account`)

**broadcast** (*signed_trx*)

Broadcast a transaction to the Steem network

**Parameters** **signed_trx** (*tx*) – Signed transaction to broadcast

**claim_reward_balance** (*reward_steem='0 STEEM'*, *reward_sbd='0 SBD'*, *reward_vests='0 VESTS'*, *account=None*)

Claim reward balances.

By default, this will claim `all` outstanding balances. To bypass this behaviour, set desired claim amount by setting any of *reward_steem*, *reward_sbd* or *reward_vests*.

**Parameters**

- **reward_steem** (*string*) – Amount of STEEM you would like to claim.
- **reward_sbd** (*string*) – Amount of SBD you would like to claim.
- **reward_vests** (*string*) – Amount of VESTS you would like to claim.
- **account** (*string*) – The source account for the claim if not `default_account` is used.

**comment_options** (*identifier*, *options*, *account=None*)

Set the comment options

**Parameters**

- **identifier** (*str*) – Post identifier
- **options** (*dict*) – The options to define.
- **account** (*str*) – (optional) the account to allow access to (defaults to `default_account`)

For the options, you have these defaults::

```
{
    "author": "",
    "permlink": "",
    "max_accepted_payout": "1000000.000 SBD",
    "percent_steem_dollars": 10000,
    "allow_votes": True,
```

```
    "allow_curation_rewards": True,
}
```

**convert**(*amount*, *account=None*, *request_id=None*)
Convert SteemDollars to Steem (takes one week to settle)

**Parameters**

- **amount** (*float*) – number of VESTS to withdraw over a period of 104 weeks
- **account** (*str*) – (optional) the source account for the transfer if not `default_account`
- **request_id** (*str*) – (optional) identifier for tracking the conversion‘

**create_account**(*account_name*, *json_meta=None*, *password=None*, *owner_key=None*, *active_key=None*, *posting_key=None*, *memo_key=None*, *additional_owner_keys=[]*, *additional_active_keys=[]*, *additional_posting_keys=[]*, *additional_owner_accounts=[]*, *additional_active_accounts=[]*, *additional_posting_accounts=[]*, *store_keys=True*, *store_owner_key=False*, *delegation_fee_steem='0 STEEM'*, *creator=None*)
Create new account in Steem

The brainkey/password can be used to recover all generated keys (see *steembase.account* for more details.

By default, this call will use `default_account` to register a new name `account_name` with all keys being derived from a new brain key that will be returned. The corresponding keys will automatically be installed in the wallet.

---

**Note:** Account creations cost a fee that is defined by the network. If you create an account, you will need to pay for that fee!

**You can partially pay that fee by delegating VESTS.**

To pay the fee in full in STEEM, leave `delegation_fee_steem` set to `0 STEEM` (Default).

To pay the fee partially in STEEM, partially with delegated VESTS, set `delegation_fee_steem` to a value greater than `1 STEEM`. *Required VESTS will be calculated automatically.*

To pay the fee with maximum amount of delegation, set `delegation_fee_steem` to `1 STEEM`. *Required VESTS will be calculated automatically.*

---

**Warning:** Don’t call this method unless you know what you are doing! Be sure to understand what this method does and where to find the private keys for your account.

---

**Note:** Please note that this imports private keys (if password is present) into the wallet by default. However, it **does not import the owner key** unless *store_owner_key* is set to True (default False). Do NOT expect to be able to recover it from the wallet if you lose your password!

---

**Parameters**

- **account_name** (*str*) – (**required**) new account name
- **json_meta** (*str*) – Optional meta data for the account
- **owner_key** (*str*) – Main owner key

- **active_key** (*str*) – Main active key
- **posting_key** (*str*) – Main posting key
- **memo_key** (*str*) – Main memo_key
- **password** (*str*) – Alternatively to providing keys, one can provide a password from which the keys will be derived
- **additional_owner_keys** (*list*) – Additional owner public keys
- **additional_active_keys** (*list*) – Additional active public keys
- **additional_posting_keys** (*list*) – Additional posting public keys
- **additional_owner_accounts** (*list*) – Additional owner account names
- **additional_active_accounts** (*list*) – Additional active account names
- **additional_posting_accounts** (*list*) – Additional posting account names
- **store_keys** (*bool*) – Store new keys in the wallet (default: `True`)
- **store_owner_key** (*bool*) – Store owner key in the wallet (default: `False`)
- **delegation_fee_steem** (*str*) – (Optional) If set, *creator* pay a fee of this amount, and delegate the rest with VESTS (calculated automatically). Minimum: 1 STEEM. If left to 0 (Default), full fee is paid without VESTS delegation.
- **creator** (*str*) – which account should pay the registration fee (defaults to `default_account`)

**Raises** **AccountExistsException** – if the account already exists on the blockchain

**custom_json**(*id*, *json*, *required_auths=[]*, *required_posting_auths=[]*)
Create a custom json operation

**Parameters**

- **id** (*str*) – identifier for the custom json (max length 32 bytes)
- **json** (*json*) – the json data to put into the custom_json operation
- **required_auths** (*list*) – (optional) required auths
- **required_posting_auths** (*list*) – (optional) posting auths

**decode_memo**(*enc_memo*)
Try to decode an encrypted memo

**delegate_vesting_shares**(*to_account: str*, *vesting_shares: str*, *account=None*)
Delegate SP to another account.

**Parameters**

- **to_account** (*string*) – Account we are delegating shares to (delegatee).
- **vesting_shares** (*string*) – Amount of VESTS to delegate eg. *10000 VESTS*.
- **account** (*string*) – The source account (delegator). If not specified, `default_account` is used.

**disallow**(*foreign*, *permission='posting'*, *account=None*, *threshold=None*)
Remove additional access to an account by some other public key or account.

**Parameters**

- **foreign** (*str*) – The foreign account that will obtain access

- **permission** (*str*) – (optional) The actual permission to modify (defaults to `posting`)
- **account** (*str*) – (optional) the account to allow access to (defaults to `default_account`)
- **threshold** (*int*) – The threshold that needs to be reached by signatures to be able to interact

**disapprove_witness** (*witness*, *account=None*)
Remove vote for a witness. This method removes a witness from your set of approved witnesses. To add witnesses see `approve_witness`.

**Parameters**

- **witness** (*str*) – witness to approve
- **account** (*str*) – (optional) the account to allow access to (defaults to `default_account`)

**finalizeOp** (*ops*, *account*, *permission*)
This method obtains the required private keys if present in the wallet, finalizes the transaction, signs it and broadacasts it

**Parameters**

- **ops** (*operation*) – The operation (or list of operaions) to broadcast
- **account** (*operation*) – The account that authorizes the operation
- **permission** (*string*) – The required permission for signing (active, owner, posting)

... note:

```
If ``ops`` is a list of operation, they all need to be
signable by the same key! Thus, you cannot combine ops
that require active permission with ops that require
posting permission. Neither can you use different
accounts for different operations!
```

**follow** (*follow, what=['blog'], account=None*)
Follow another account's blog

**Parameters**

- **follow** (*str*) – Follow this account
- **what** (*list*) – List of states to follow (defaults to `['blog']`)
- **account** (*str*) – (optional) the account to allow access to (defaults to `default_account`)

**interest** (*account*)
Caluclate interest for an account

**Parameters** **account** (*str*) – Account name to get interest for

**post** (*title*, *body*, *author*, *permlink=None*, *reply_identifier=None*, *json_metadata=None*, *comment_options=None*, *community=None*, *tags=None*, *beneficiaries=None*, *self_vote=False*)
Create a new post.

If this post is intended as a reply/comment, *reply_identifier* needs to be set with the identifier of the parent post/comment (eg. *@author/permlink*).

Optionally you can also set json_metadata, comment_options and upvote the newly created post as an author.

Setting category, tags or community will override the values provided in json_metadata and/or comment_options where appropriate.

**Parameters**

- **title** (*str*) – Title of the post
- **body** (*str*) – Body of the post/comment
- **author** (*str*) – Account are you posting from
- **permlink** (*str*) – Manually set the permlink (defaults to None). If left empty, it will be derived from title automatically.
- **reply_identifier** (*str*) – Identifier of the parent post/comment (only if this post is a reply/comment).
- **json_metadata** (*str, dict*) – JSON meta object that can be attached to the post.
- **comment_options** (*str, dict*) – JSON options object that can be attached to the post. Example:

```
comment_options = {
    'max_accepted_payout': '1000000.000 SBD',
    'percent_steem_dollars': 10000,
    'allow_votes': True,
    'allow_curation_rewards': True,
    'extensions': [[0, {
        'beneficiaries': [
            {'account': 'account1', 'weight': 5000},
            {'account': 'account2', 'weight': 5000},
        ]}
    ]]
}
```

- **community** (*str*) – (Optional) Name of the community we are posting into. This will also override the community specified in *json_metadata*.
- **tags** (*str, list*) – (Optional) A list of tags (5 max) to go with the post. This will also override the tags specified in *json_metadata*. The first tag will be used as a 'category'. If provided as a string, it should be space separated.
- **beneficiaries** (*list of dicts*) – (Optional) A list of beneficiaries for posting reward distribution. This argument overrides beneficiaries as specified in *comment_options*.

  For example, if we would like to split rewards between account1 and account2:

```
beneficiaries = [
    {'account': 'account1', 'weight': 5000},
    {'account': 'account2', 'weight': 5000}
]
```

- **self_vote** (*bool*) – (Optional) Upvote the post as author, right after posting.

**resteem**(*identifier*, *account=None*)

Resteem a post

**Parameters**

- **identifier** (*str*) – post identifier (@<account>/<permlink>)

- **account** (*str*) – (optional) the account to allow access to (defaults to `default_account`)

**set_withdraw_vesting_route**(*to*, *percentage=100*, *account=None*, *auto_vest=False*)
Set up a vesting withdraw route. When vesting shares are withdrawn, they will be routed to these accounts based on the specified weights.

**Parameters**

- **to** (*str*) – Recipient of the vesting withdrawal
- **percentage** (*float*) – The percent of the withdraw to go to the 'to' account.
- **account** (*str*) – (optional) the vesting account
- **auto_vest** (*bool*) – Set to true if the from account should receive the VESTS as VESTS, or false if it should receive them as STEEM. (defaults to `False`)

**sign**(*unsigned_trx*, *wifs=[]*)
Sign a provided transaction witht he provided key(s)

**Parameters**

- **unsigned_trx** (*dict*) – The transaction to be signed and returned
- **wifs** (*string*) – One or many wif keys to use for signing a transaction. If not present, the keys will be loaded from the wallet as defined in "missing_signatures" key of the transactizions.

**transfer**(*to*, *amount*, *asset*, *memo=''*, *account=None*)
Transfer SBD or STEEM to another account.

**Parameters**

- **to** (*str*) – Recipient
- **amount** (*float*) – Amount to transfer
- **asset** (*str*) – Asset to transfer (`SBD` or `STEEM`)
- **memo** (*str*) – (optional) Memo, may begin with # for encrypted messaging
- **account** (*str*) – (optional) the source account for the transfer if not `default_account`

**transfer_from_savings**(*amount*, *asset*, *memo*, *request_id=None*, *to=None*, *account=None*)
Withdraw SBD or STEEM from 'savings' account.

**Parameters**

- **amount** (*float*) – STEEM or SBD amount
- **asset** (*float*) – 'STEEM' or 'SBD'
- **memo** (*str*) – (optional) Memo
- **request_id** (*str*) – (optional) identifier for tracking or cancelling the withdrawal
- **to** (*str*) – (optional) the source account for the transfer if not `default_account`
- **account** (*str*) – (optional) the source account for the transfer if not `default_account`

**transfer_from_savings_cancel**(*request_id*, *account=None*)
Cancel a withdrawal from 'savings' account.

**Parameters**

- **request_id** (*str*) – Identifier for tracking or cancelling the withdrawal
- **account** (*str*) – (optional) the source account for the transfer if not `default_account`

**transfer_to_savings**(*amount*, *asset*, *memo*, *to=None*, *account=None*)
Transfer SBD or STEEM into a 'savings' account.

**Parameters**

- **amount** (*float*) – STEEM or SBD amount
- **asset** (*float*) – 'STEEM' or 'SBD'
- **memo** (*str*) – (optional) Memo
- **to** (*str*) – (optional) the source account for the transfer if not `default_account`
- **account** (*str*) – (optional) the source account for the transfer if not `default_account`

**transfer_to_vesting**(*amount*, *to=None*, *account=None*)
Vest STEEM

**Parameters**

- **amount** (*float*) – number of STEEM to vest
- **to** (*str*) – (optional) the source account for the transfer if not `default_account`
- **account** (*str*) – (optional) the source account for the transfer if not `default_account`

**unfollow**(*unfollow, what=['blog'], account=None*)
Unfollow another account's blog

**Parameters**

- **unfollow** (*str*) – Follow this account
- **what** (*list*) – List of states to follow (defaults to `['blog']`)
- **account** (*str*) – (optional) the account to allow access to (defaults to `default_account`)

**update_account_profile**(*profile*, *account=None*)
Update an account's meta data (json_meta)

**Parameters**

- **json** (*dict*) – The meta data to use (i.e. use Profile() from account.py)
- **account** (*str*) – (optional) the account to allow access to (defaults to `default_account`)

**update_memo_key**(*key*, *account=None*)
Update an account's memo public key

This method does **not** add any private keys to your wallet but merely changes the memo public key.

**Parameters**

- **key** (*str*) – New memo public key
- **account** (*str*) – (optional) the account to allow access to (defaults to `default_account`)

**vote**(*identifier*, *weight*, *account=None*)
Vote for a post

**Parameters**

- **identifier** (*str*) – Identifier for the post to upvote Takes the form `@author/permlink`
- **weight** (*float*) – Voting weight. Range: -100.0 - +100.0. May not be 0.0
- **account** (*str*) – Voter to use for voting. (Optional)

If `voter` is not defines, the `default_account` will be taken or a ValueError will be raised

```
steempy set default_account <account>
```

**withdraw_vesting**(*amount*, *account=None*)
Withdraw VESTS from the vesting account.

**Parameters**

- **amount** (*float*) – number of VESTS to withdraw over a period of 104 weeks
- **account** (*str*) – (optional) the source account for the transfer if not `default_account`

**witness_feed_publish**(*steem_usd_price*, *quote='1.000'*, *account=None*)
Publish a feed price as a witness.

**Parameters**

- **steem_usd_price** (*float*) – Price of STEEM in USD (implied price)
- **quote** (*float*) – (optional) Quote Price. Should be 1.000, unless we are adjusting the feed to support the peg.
- **account** (*str*) – (optional) the source account for the transfer if not `default_account`

**witness_update**(*signing_key*, *url*, *props*, *account=None*)
Update witness

**Parameters**

- **signing_key** (*pubkey*) – Signing key
- **url** (*str*) – URL
- **props** (*dict*) – Properties
- **account** (*str*) – (optional) witness account name

Properties::

```
{
    "account_creation_fee": x,
    "maximum_block_size": x,
    "sbd_interest_rate": x,
}
```

## TransactionBuilder

**class** steem.transactionbuilder.**TransactionBuilder**(*tx=None*, *steemd_instance=None*, *wallet_instance=None*, *no_broadcast=False*, *expiration=60*)

This class simplifies the creation of transactions by adding operations and signers.

**addSigningInformation**(*account*, *permission*)

This is a private method that adds side information to a unsigned/partial transaction in order to simplify later signing (e.g. for multisig or coldstorage)

**broadcast**()

Broadcast a transaction to the Steem network

**Parameters** **tx** (*tx*) – Signed transaction to broadcast

**sign**()

Sign a provided transaction witht he provided key(s)

**Parameters**

- **tx** (*dict*) – The transaction to be signed and returned
- **wifs** (*string*) – One or many wif keys to use for signing a transaction. If not present, the keys will be loaded from the wallet as defined in "missing_signatures" key of the transactions.

---

## Wallet

Wallet is a low-level utility. It could be used to create 3rd party cli and GUI wallets on top of `steem-python`'s infrastructure.

**class** steem.wallet.**Wallet**(*steemd_instance=None*, ***kwargs*)

The wallet is meant to maintain access to private keys for your accounts. It either uses manually provided private keys or uses a SQLite database managed by storage.py.

**Parameters**

- **rpc** (Steem) – RPC connection to a Steem node
- **keys** (*array,dict,string*) – Predefine the wif keys to shortcut the wallet database

Three wallet operation modes are possible:

•**Wallet Database**: Here, steemlibs loads the keys from the locally stored wallet SQLite database (see `storage.py`). To use this mode, simply call `Steem()` without the `keys` parameter

•**Providing Keys**: Here, you can provide the keys for your accounts manually. All you need to do is add the wif keys for the accounts you want to use as a simple array using the `keys` parameter to `Steem()`.

•**Force keys**: This more is for advanced users and requires that you know what you are doing. Here, the `keys` parameter is a dictionary that overwrite the `active`, `owner`, `posting` or `memo` keys for any account. This mode is only used for *foreign* signatures!

**addPrivateKey**(*wif*)

Add a private key to the wallet database

**changePassphrase**()

Change the passphrase for the wallet database

**created**()
Do we have a wallet database already?

**decrypt_wif**(*encwif*)
decrypt a wif key

**encrypt_wif**(*wif*)
Encrypt a wif key

**getAccount**(*pub*)
Get the account data for a public key

**getAccountFromPrivateKey**(*wif*)
Obtain account name from private key

**getAccountFromPublicKey**(*pub*)
Obtain account name from public key

**getAccounts**()
Return all accounts installed in the wallet database

**getAccountsWithPermissions**()
Return a dictionary for all installed accounts with their corresponding installed permissions

**getActiveKeyForAccount**(*name*)
Obtain owner Active Key for an account from the wallet database

**getKeyType**(*account*, *pub*)
Get key type

**getMemoKeyForAccount**(*name*)
Obtain owner Memo Key for an account from the wallet database

**getOwnerKeyForAccount**(*name*)
Obtain owner Private Key for an account from the wallet database

**getPassword**(*confirm=False*, *text='Passphrase: '*)
Obtain a password from the user

**getPostingKeyForAccount**(*name*)
Obtain owner Posting Key for an account from the wallet database

**getPrivateKeyForPublicKey**(*pub*)
Obtain the private key for a given public key

**Parameters** **pub** (*str*) – Public Key

**getPublicKeys**()
Return all installed public keys

**lock**()
Lock the wallet database

**locked**()
Is the wallet database locked?

**newWallet**()
Create a new wallet database

**removeAccount**(*account*)
Remove all keys associated with a given account

**removePrivateKeyFromPublicKey**(*pub*)
Remove a key from the wallet database

**setKeys** (*loadkeys*)
This method is strictly only for in memory keys that are passed to Wallet/Steem with the `keys` argument

**unlock** (*pwd=None*)
Unlock the wallet database

# Tools

`steem-python` comes with batteries included.

This page lists a collection of convenient tools at your disposal.

## Account

**class** `steem.account.`**Account** (*account_name*, *steemd_instance=None*)
This class allows to easily access Account data

**Parameters**

- **account_name** (*str*) – Name of the account
- **steemd_instance** (Steemd) – Steemd() instance to use when accessing a RPC

**export** (*load_extras=True*)
This method returns a dictionary that is type-safe to store as JSON or in a database.

**Parameters** **load_extras** (*bool*) – Fetch extra information related to the account (this might take a while).

**get_account_history** (*index*, *limit*, *start=None*, *stop=None*, *order=-1*, *filter_by=None*, *raw_output=False*)
A generator over steemd.get_account_history.

It offers serialization, filtering and fine grained iteration control.

**Parameters**

- **index** (*int*) – start index for get_account_history
- **limit** (*int*) – How many items are we interested in.
- **start** (*int*) – (Optional) skip items until this index
- **stop** (*int*) – (Optional) stop iteration early at this index
- **order** – (1, -1): 1 for chronological, -1 for reverse order
- **filter_by** (*str, list*) – filter out all but these operations
- **raw_output** (*bool*) – (Defaults to False). If True, return history in steemd format (unchanged).

**history** (*filter_by=None*, *start=0*, *batch_size=1000*, *raw_output=False*)
Stream account history in chronological order.

**history_reverse** (*filter_by=None*, *batch_size=1000*, *raw_output=False*)
Stream account history in reverse chronological order.

## Amount

**class** steem.amount.**Amount**(*amount_string='0 SBD'*)
This class helps deal and calculate with the different assets on the chain.

**Parameters** **amountString** (*str*) – Amount string as used by the backend (e.g. "10 SBD")

---

## Blockchain

**class** steem.blockchain.**Blockchain**(*steemd_instance=None*, *mode='irreversible'*)
Access the blockchain and read data from it.

**Parameters**

- **steemd_instance** (Steemd) – Steemd() instance to use when accessing a RPC
- **mode** (*str*) – *irreversible* or *head*. *irreversible* is default.

**get_all_usernames**(**args*, ***kwargs*)
Fetch the full list of STEEM usernames.

**get_current_block**()
This call returns the current block

**get_current_block_num**()
This call returns the current block

**static hash_op**(*event: dict*)
This method generates a hash of blockchain operation.

**history**(*filter_by: typing.Union[str, list] = []*, *start_block=1*, *end_block=None*, *raw_output=False*, ***kwargs*)
Yield a stream of historic operations.

Similar to Blockchain.stream(), but starts at beginning of chain unless start_block is set.

**Parameters**

- **filter_by** (*str, list*) – List of operations to filter for
- **start_block** (*int*) – Block to start with. If not provided, start of blockchain is used (block 1).
- **end_block** (*int*) – Stop iterating at this block. If not provided, this generator will run forever.
- **raw_output** (*bool*) – (Defaults to False). If True, return ops in a unmodified steemd structure.

**info**()
This call returns the *dynamic global properties*

**stream**(*filter_by: typing.Union[str, list] = []*, **args*, ***kwargs*)
Yield a stream of operations, starting with current head block.

**Parameters** **filter_by** (*str, list*) – List of operations to filter for

**stream_from**(*start_block=None*, *end_block=None*, *batch_operations=False*, *full_blocks=False*, ***kwargs*)
This call yields raw blocks or operations depending on full_blocks param.

By default, this generator will yield operations, one by one. You can choose to yield lists of operations, batched to contain all operations for each block with `batch_operations=True`. You can also yield full blocks instead, with `full_blocks=True`.

**Parameters**

- **start_block** (*int*) – Block to start with. If not provided, current (head) block is used.
- **end_block** (*int*) – Stop iterating at this block. If not provided, this generator will run forever (streaming mode).
- **batch_operations** (*bool*) – (Defaults to False) Rather than yielding operations one by one, yield a list of all operations for each block.
- **full_blocks** (*bool*) – (Defaults to False) Rather than yielding operations, return raw, unedited blocks as provided by steemd. This mode will NOT include virtual operations.

---

## Blog

**class** steem.blog.**Blog**(*account_name: str*, *comments_only=False*, *steemd_instance=None*)

Obtain a list of blog posts for an account

**Parameters**

- **account_name** (*str*) – Name of the account
- **comments_only** (*bool*) – (Default False). Toggle between posts and comments.
- **steemd_instance** (Steemd) – Steemd instance overload

**Returns** Generator with Post objects in reverse chronological order.

### Example

To get all posts, you can use either generator:

```
gen1 = Blog('furion')
gen2 = b.all()

next(gen1)
next(gen2)
```

To get some posts, you can call *take()*:

```
b = Blog('furion')
posts = b.take(5)
```

**all**()

A generator that will return ALL of account history.

**take**(*limit=5*)

Take up to n (n = limit) posts/comments at a time.

You can call this method as many times as you want. Once there are no more posts to take, it will return [].

**Returns** List of posts/comments in a batch of size up to *limit*.

---

## Converter

**class** steem.converter.**Converter**(*steemd_instance=None*)
Converter simplifies the handling of different metrics of the blockchain

**Parameters** **steemd_instance** (Steemd) – Steemd() instance to use when accessing a RPC

**rshares_2_weight**(*rshares*)
Obtain weight from rshares

**Parameters** **rshares** (*number*) – R-Shares

**sbd_median_price**()
Obtain the sbd price as derived from the median over all witness feeds. Return value will be SBD

**sbd_to_rshares**(*sbd_payout*)
Obtain r-shares from SBD

**Parameters** **sbd_payout** (*number*) – Amount of SBD

**sbd_to_steem**(*amount_sbd*)
Conversion Ratio for given amount of SBD to STEEM at current price feed

**Parameters** **amount_sbd** (*number*) – Amount of SBD

**sp_to_rshares**(*sp*, *voting_power=10000*, *vote_pct=10000*)
Obtain the r-shares

**Parameters**

- **sp** (*number*) – Steem Power
- **voting_power** (*int*) – voting power (100% = 10000)
- **vote_pct** (*int*) – voting participation (100% = 10000)

**sp_to_vests**(*sp*)
Obtain VESTS (not MVESTS!) from SP

**Parameters** **sp** (*number*) – SP to convert

**steem_per_mvests**()
Obtain STEEM/MVESTS ratio

**steem_to_sbd**(*amount_steem*)
Conversion Ratio for given amount of STEEM to SBD at current price feed

**Parameters** **amount_steem** (*number*) – Amount of STEEM

**vests_to_sp**(*vests*)
Obtain SP from VESTS (not MVESTS!)

**Parameters** **vests** (*number*) – Vests to convert to SP

## Dex

**class** steem.dex.**Dex**(*steemd_instance=None*)
This class allows to access calls specific for the internal exchange of STEEM.

**Parameters** **steemd_instance** (Steemd) – Steemd() instance to use when accessing a RPC

**buy**(*amount*, *quote_symbol*, *rate*, *expiration=604800*, *killfill=False*, *account=None*, *order_id=None*)
Places a buy order in a given market (buy `quote`, sell `base` in market `quote_base`). If successful, the method will return the order creating (signed) transaction.

**Parameters**

- **amount** (*number*) – Amount of `quote` to buy
- **quote_symbol** (*str*) – STEEM, or SBD
- **price** (*float*) – price denoted in `base`/`quote`
- **expiration** (*number*) – (optional) expiration time of the order in seconds (defaults to 7 days)
- **killfill** (*bool*) – flag that indicates if the order shall be killed if it is not filled (defaults to False)
- **account** (*str*) – (optional) the source account for the transfer if not `default_account`
- **order_id** (*int*) – (optional) a 32bit orderid for tracking of the created order (random by default)

Prices/Rates are denoted in 'base', i.e. the STEEM:SBD market is priced in SBD per STEEM.

**cancel**(*orderid*, *account=None*)
Cancels an order you have placed in a given market.

**Parameters**

- **orderid** (*int*) – the 32bit orderid
- **account** (*str*) – (optional) the source account for the transfer if not `default_account`

**get_ticker**()
Returns the ticker for all markets.

Output Parameters:

- `latest`: Price of the order last filled
- `lowest_ask`: Price of the lowest ask
- `highest_bid`: Price of the highest bid
- `sbd_volume`: Volume of SBD
- `steem_volume`: Volume of STEEM
- `percent_change`: 24h change percentage (in %)

---

**Note:** Market is STEEM:SBD and prices are SBD per STEEM!

---

Sample Output:

```
{'highest_bid': 0.30100226633322913,
 'latest': 0.0,
 'lowest_ask': 0.3249636958897082,
 'percent_change': 0.0,
 'sbd_volume': 108329611.0,
 'steem_volume': 355094043.0}
```

**market_history**(*bucket_seconds=300*, *start_age=3600*, *end_age=0*)
Return the market history (filled orders).

**Parameters**

- **bucket_seconds** (*int*) – Bucket size in seconds (see *returnMarketHistoryBuckets()*)
- **start_age** (*int*) – Age (in seconds) of the start of the window (default: 1h/3600)
- **end_age** (*int*) – Age (in seconds) of the end of the window (default: now/0)

Example:

```
{'close_sbd': 2493387,
 'close_steem': 7743431,
 'high_sbd': 1943872,
 'high_steem': 5999610,
 'id': '7.1.5252',
 'low_sbd': 534928,
 'low_steem': 1661266,
 'open': '2016-07-08T11:25:00',
 'open_sbd': 534928,
 'open_steem': 1661266,
 'sbd_volume': 9714435,
 'seconds': 300,
 'steem_volume': 30088443},
```

**sell**(*amount*, *quote_symbol*, *rate*, *expiration=604800*, *killfill=False*, *account=None*, *orderid=None*)
Places a sell order in a given market (sell `quote`, buy `base` in market `quote_base`). If successful, the method will return the order creating (signed) transaction.

**Parameters**

- **amount** (*number*) – Amount of `quote` to sell
- **quote_symbol** (*str*) – STEEM, or SBD
- **price** (*float*) – price denoted in `base`/`quote`
- **expiration** (*number*) – (optional) expiration time of the order in seconds (defaults to 7 days)
- **killfill** (*bool*) – flag that indicates if the order shall be killed if it is not filled (defaults to False)
- **account** (*str*) – (optional) the source account for the transfer if not `default_account`
- **orderid** (*int*) – (optional) a 32bit orderid for tracking of the created order (random by default)

Prices/Rates are denoted in 'base', i.e. the STEEM:SBD market is priced in SBD per STEEM.

**trade_history**(*time=3600*, *limit=100*)
Returns the trade history for the internal market

**Parameters**

- **time** (*int*) – Show the last x seconds of trades (default 1h)
- **limit** (*int*) – amount of trades to show (<100) (default: 100)

## Post

**class** steem.post.**Post**(*post*, *steemd_instance=None*)
This object gets instantiated by Steem.streams and is used as an abstraction layer for Comments in Steem

**Parameters**

- **post** (*str or dict*) – @author/permlink or raw comment as dictionary.
- **steemd_instance** (Steemd) – Steemd node to connect to

**curation_reward_pct**()
If post is less than 30 minutes old, it will incur a curation reward penalty.

**downvote**(*weight=-100*, *voter=None*)
Downvote the post

**Parameters**

- **weight** (*float*) – (optional) Weight for posting (-100.0 - +100.0) defaults to -100.0
- **voter** (*str*) – (optional) Voting account

**edit**(*body*, *meta=None*, *replace=False*)
Edit an existing post

**Parameters**

- **body** (*str*) – Body of the reply
- **meta** (*json*) – JSON meta object that can be attached to the post. (optional)
- **replace** (*bool*) – Instead of calculating a *diff*, replace the post entirely (defaults to False)

**export**()
This method returns a dictionary that is type-safe to store as JSON or in a database.

**static get_all_replies**(*root_post=None*, *comments=[]*, *all_comments=[]*)
Recursively fetch all the child comments, and return them as a list.

Usage: all_comments = Post.get_all_replies(Post('@foo/bar'))

**get_replies**()
Return **first-level** comments of the post.

**is_comment**()
Retuns True if post is a comment

**is_main_post**()
Retuns True if main post, and False if this is a comment (reply).

**static parse_identifier**(*uri*)
Extract post identifier from post URL.

**reply**(*body*, *title=''*, *author=''*, *meta=None*)
Reply to an existing post

**Parameters**

- **body** (*str*) – Body of the reply
- **title** (*str*) – Title of the reply post
- **author** (*str*) – Author of reply (optional) if not provided default_user will be used, if present, else a ValueError will be raised.

- **meta** (*json*) – JSON meta object that can be attached to the post. (optional)

**reward**
Return a float value of estimated total SBD reward.

**time_elapsed**()
Return a timedelta on how old the post is.

**upvote**(*weight=100*, *voter=None*)
Upvote the post

**Parameters**

- **weight** (*float*) – (optional) Weight for posting (-100.0 - +100.0) defaults to +100.0
- **voter** (*str*) – (optional) Voting account

**vote**(*weight*, *voter=None*)
Vote the post

**Parameters**

- **weight** (*float*) – Weight for posting (-100.0 - +100.0)
- **voter** (*str*) – Voting account

---

## steem.utils

steem.utils.**block_num_from_hash**(*block_hash: str*) → int
return the first 4 bytes (8 hex digits) of the block ID (the block_num) :param block_hash: :type block_hash: str

**Returns**

**Return type** int

steem.utils.**block_num_from_previous**(*previous_block_hash: str*) → int

**Parameters** **previous_block_hash** (*str*) –

**Returns**

**Return type** int

steem.utils.**chunkify**(*iterable*, *chunksize=10000*)
Yield successive chunksized chunks from iterable.

**Parameters**

- **iterable** –
- **chunksize** – (Default value = 10000)

Returns:

steem.utils.**construct_identifier**(**args*, *username_prefix='@'*)
Create a post identifier from comment/post object or arguments.

#### Examples

```
construct_identifier('username', 'permlink')
construct_identifier({'author': 'username', 'permlink': 'permlink'})
```

steem.utils.**env_unlocked**()
Check if wallet password is provided as ENV variable.

steem.utils.**fmt_time**(*t*)
Properly Format Time for permlinks

steem.utils.**fmt_time_from_now**(*secs=0*)
Properly Format Time that is *x* seconds in the future

**Parameters** **secs** (*int*) – Seconds to go in the future (*x>0*) or the past (*x<0*)

**Returns** Properly formated time for Graphene (*%Y-%m-%dT%H:%M:%S*)

**Return type** str

steem.utils.**fmt_time_string**(*t*)
Properly Format Time for permlinks

steem.utils.**is_comment**(*item*)
Quick check whether an item is a comment (reply) to another post. The item can be a Post object or just a raw comment object from the blockchain.

steem.utils.**json_expand**(*json_op*, *key_name='json'*)
Convert a string json object to Python dict in an op.

steem.utils.**keep_in_dict**(*obj*, *allowed_keys=[]*)
Prune a class or dictionary of all but allowed keys.

steem.utils.**parse_time**(*block_time*)
Take a string representation of time from the blockchain, and parse it into datetime object.

steem.utils.**remove_from_dict**(*obj*, *remove_keys=[]*)
Prune a class or dictionary of specified keys.

steem.utils.**strfage**(*time*, *fmt=None*)
Format time/age

steem.utils.**strfdelta**(*tdelta*, *fmt*)
Format time/age

steem.utils.**time_elapsed**(*posting_time*)
Takes a string time from a post or blockchain event, and returns a time delta from now.

---

# Low Level

## HttpClient

A fast `urllib3` based HTTP client that features:

- Connection Pooling
- Concurrent Processing

- Automatic Node Failover

The functionality of `HttpClient` is encapsulated by `Steem` class. You shouldn't be using `HttpClient` directly, unless you know exactly what you're doing.

**class** `steembase.http_client.`**`HttpClient`**(*nodes*, ***kwargs*)
Simple Steem JSON-HTTP-RPC API

This class serves as an abstraction layer for easy use of the Steem API.

**Parameters** **`nodes`** (*`list`*) – A list of Steem HTTP RPC nodes to connect to.

```
from steem.http_client import HttpClient
rpc = HttpClient(['https://steemd-node1.com', 'https://steemd-node2.com'])
```

any call available to that port can be issued using the instance via the syntax `rpc.exec('command', *parameters)`.

Example:

```
rpc.exec(
    'get_followers',
    'furion', 'abit', 'blog', 10,
    api='follow_api'
)
```

**`exec`**(*name*, **args*, *api=None*, *return_with_args=None*, *_ret_cnt=0*)
Execute a method against steemd RPC.

> **Warning:** This command will auto-retry in case of node failure, as well as handle node fail-over, unless we are broadcasting a transaction. In latter case, the exception is **re-raised**.

**static** **`json_rpc_body`**(*name*, **args*, *api=None*, *as_json=True*, *_id=0*)
Build request body for steemd RPC requests.

**Parameters**

- **`name`** (*`str`*) – Name of a method we are trying to call. (ie: *get_accounts*)
- **`args`** – A list of arguments belonging to the calling method.
- **`api`** (*`None, str`*) – If api is provided (ie: *follow_api*), we generate a body that uses *call* method appropriately.
- **`as_json`** (*`bool`*) – Should this function return json as dictionary or string.
- **`_id`** (*`int`*) – This is an arbitrary number that can be used for request/response tracking in multi-threaded scenarios.

**Returns** If *as_json* is set to *True*, we get json formatted as a string. Otherwise, a Python dictionary is returned.

**Return type** (dict,str)

**`next_node`**()
Switch to the next available node.

This method will change base URL of our requests. Use it when the current node goes down to change to a fallback node.

**`set_node`**(*node_url*)
Change current node to provided node URL.

## steembase

SteemBase contains various primitives for building higher level abstractions. This module should only be used by library developers or people with deep domain knowledge.

**Warning:** Not all methods are documented. Please see source.

### Account

**class** steembase.account.**Address**(*address=None*, *pubkey=None*, *prefix='STM'*)
Address class

This class serves as an address representation for Public Keys.

**Parameters**

- **address** (*str*) – Base58 encoded address (defaults to None)
- **pubkey** (*str*) – Base58 encoded pubkey (defaults to None)
- **prefix** (*str*) – Network prefix (defaults to GPH)

Example:

```
Address("GPHFN9r6VYzBK8EKtMewfNbfiGCr56pHDBFi")
```

**derivesha256address**()
Derive address using RIPEMD160(SHA256(x))

**derivesha512address**()
Derive address using RIPEMD160(SHA512(x))

**class** steembase.account.**BrainKey**(*brainkey=None*, *sequence=0*)
Brainkey implementation similar to the graphene-ui web-wallet.

**Parameters**

- **brainkey** (*str*) – Brain Key
- **sequence** (*int*) – Sequence number for consecutive keys

Keys in Graphene are derived from a seed brain key which is a string of 16 words out of a predefined dictionary with 49744 words. It is a simple single-chain key derivation scheme that is not compatible with BIP44 but easy to use.

Given the brain key, a private key is derived as:

```
privkey = SHA256(SHA512(brainkey + " " + sequence))
```

Incrementing the sequence number yields a new key that can be regenerated given the brain key.

**get_brainkey**()
Return brain key of this instance

**get_private**()
Derive private key from the brain key and the current sequence number

**next_sequence**()
Increment the sequence number by 1

**normalize**(*brainkey*)
Correct formating with single whitespace syntax and no trailing space

**suggest**()
Suggest a new random brain key. Randomness is provided by the operating system using `os.urandom()`.

**class** steembase.account.**PasswordKey**(*account*, *password*, *role='active'*)
This class derives a private key given the account name, the role and a password. It leverages the technology of Brainkeys and allows people to have a secure private key by providing a passphrase only.

**get_private**()
Derive private key from the brain key and the current sequence number

**class** steembase.account.**PrivateKey**(*wif=None*, *prefix='STM'*)
Derives the compressed and uncompressed public keys and constructs two instances of `PublicKey`:

**Parameters**

- **wif** (*str*) – Base58check-encoded wif key
- **prefix** (*str*) – Network prefix (defaults to `GPH`)

Example::

```
PrivateKey("5HqUkGuo62BfcJU5vNhTXKJRXuUi9QSE6jp8C3uBJ2BVHtB8WSd")
```

Compressed vs. Uncompressed:

- **`PrivateKey("w-i-f").pubkey`:** Instance of `PublicKey` using compressed key.
- **`PrivateKey("w-i-f").pubkey.address`:** Instance of `Address` using compressed key.
- **`PrivateKey("w-i-f").uncompressed`:** Instance of `PublicKey` using uncompressed key.
- **`PrivateKey("w-i-f").uncompressed.address`:** Instance of `Address` using uncompressed key.

**compressedpubkey**()
Derive uncompressed public key

**class** steembase.account.**PublicKey**(*pk*, *prefix='STM'*)
This class deals with Public Keys and inherits `Address`.

**Parameters**

- **pk** (*str*) – Base58 encoded public key
- **prefix** (*str*) – Network prefix (defaults to `GPH`)

Example::

```
PublicKey("GPH6UtYWWs3rkZGV8JA86qrgkG6tyFksgECefKE1MiH4HkLD8PFGL")
```

---

**Note:** By default, graphene-based networks deal with **compressed** public keys. If an **uncompressed** key is required, the method `unCompressed` can be used:

```
PublicKey("xxxxx").unCompressed()
```

---

**compressed**()
Derive compressed public key

**point**()
Return the point for the public key

**unCompressed**()
Derive uncompressed key

---

### Base58

**class** steembase.base58.**Base58**(*data*, *prefix='STM'*)
Base58 base class

This class serves as an abstraction layer to deal with base58 encoded strings and their corresponding hex and binary representation throughout the library.

**Parameters**

- **data** (*hex, wif, bip38 encrypted wif, base58 string*) – Data to initialize object, e.g. pubkey data, address data, ...
- **prefix** (*str*) – Prefix to use for Address/PubKey strings (defaults to GPH)

**Returns** Base58 object initialized with data

**Return type** *Base58*

**Raises** **ValueError** – if data cannot be decoded

- bytes(Base58): Returns the raw data
- str(Base58): Returns the readable Base58CheckEncoded data.
- repr(Base58): Gives the hex representation of the data.
- **format(Base58, _format) Formats the instance according to _format:**
  - "btc": prefixed with 0x80. Yields a valid btc address
  - "wif": prefixed with 0x00. Yields a valid wif key
  - "bts": prefixed with BTS
  - etc.

---

### Bip38

steembase.bip38.**decrypt**(*encrypted_privkey*, *passphrase*)
BIP0038 non-ec-multiply decryption. Returns WIF privkey.

**Parameters**

- **encrypted_privkey** (Base58) – Private key
- **passphrase** (*str*) – UTF-8 encoded passphrase for decryption

**Returns** BIP0038 non-ec-multiply decrypted key

**Return type** *Base58*

**Raises** **SaltException** – if checksum verification failed (e.g. wrong password)

steembase.bip38.**encrypt**(*privkey*, *passphrase*)
BIP0038 non-ec-multiply encryption. Returns BIP0038 encrypted privkey.

**Parameters**

- **privkey** (Base58) – Private key
- **passphrase** (*str*) – UTF-8 encoded passphrase for encryption

**Returns** BIP0038 non-ec-multiply encrypted wif key

**Return type** *Base58*

---

### Memo

steembase.memo.**decode_memo**(*priv*, *message*)
Decode a message with a shared secret between Alice and Bob

**Parameters**

- **priv** (PrivateKey) – Private Key (of Bob)
- **message** (*base58encoded*) – Encrypted Memo message

**Returns** Decrypted message

**Return type** str

**Raises** **ValueError** – if message cannot be decoded as valid UTF-8 string

steembase.memo.**encode_memo**(*priv*, *pub*, *nonce*, *message*, ***kwargs*)
Encode a message with a shared secret between Alice and Bob

**Parameters**

- **priv** (PrivateKey) – Private Key (of Alice)
- **pub** (PublicKey) – Public Key (of Bob)
- **nonce** (*int*) – Random nonce
- **message** (*str*) – Memo message

**Returns** Encrypted message

**Return type** hex

steembase.memo.**get_shared_secret**(*priv*, *pub*)
Derive the share secret between `priv` and `pub`

**Parameters**

- **priv** (Base58) – Private Key
- **pub** (Base58) – Public Key

**Returns** Shared secret

**Return type** hex

The shared secret is generated such that:

```
Pub(Alice) * Priv(Bob) = Pub(Bob) * Priv(Alice)
```

steembase.memo.**init_aes**(*shared_secret*, *nonce*)
Initialize AES instance

**Parameters**

- **shared_secret** (*hex*) – Shared Secret to use as encryption key
- **nonce** (*int*) – Random nonce

**Returns** AES instance and checksum of the encryption key

**Return type** length 2 tuple

steembase.memo.**involved_keys**(*message*)
decode structure

---

### Operations

**class** steembase.operations.**CommentOptionExtensions**(*o*)
Serialize Comment Payout Beneficiaries.

**Parameters** **beneficiaries** (*list*) – A static_variant containing beneficiaries.

#### Example

```
[0,
    {'beneficiaries': [
        {'account': 'furion', 'weight': 10000}
    ]}
]
```

**class** steembase.operations.**GrapheneObject**(*data=None*)
Core abstraction class

This class is used for any JSON reflected object in Graphene.

- `instance.__json__()`: encodes data into json format
- `bytes(instance)`: encodes data into wire format
- `str(instances)`: dumps json object as string

---

### Transactions

**class** steembase.transactions.**SignedTransaction**(**args*, ***kwargs*)
Create a signed transaction and offer method to create the signature

**Parameters**

- **refNum** (*num*) – parameter ref_block_num (see `getBlockParams`)
- **refPrefix** (*num*) – parameter ref_block_prefix (see `getBlockParams`)
- **expiration** (*str*) – expiration date
- **operations** (*Array*) – array of operations

**derSigToHexSig**(*s*)
Format DER to HEX signature

**recoverPubkeyParameter**(*digest*, *signature*, *pubkey*)
Use to derive a number that allows to easily recover the public key from the signature

**recover_public_key**(*digest*, *signature*, *i*)
Recover the public key from the the signature

**sign**(*wifkeys*, *chain=None*)
Sign the transaction with the provided private keys.

**Parameters**

- **wifkeys** (*list*) – Array of wif keys
- **chain** (*str*) – identifier for the chain

steembase.transactions.**fmt_time_from_now**(*secs=0*)
Properly Format Time that is *x* seconds in the future

**Parameters** **secs** (*int*) – Seconds to go in the future (*x>0*) or the past (*x<0*)

**Returns** Properly formated time for Graphene (*%Y-%m-%dT%H:%M:%S*)

**Return type** str

steembase.transactions.**get_block_params**(*steem*)
Auxiliary method to obtain `ref_block_num` and `ref_block_prefix`. Requires a websocket connection to a witness node!

---

## Types

steembase.types.**JsonObj**(*data*)
Returns json object from data

**class** steembase.types.**ObjectId**(*object_str*, *type_verify=None*)
Encodes object/protocol ids

steembase.types.**variable_buffer**(*s*)
Encode variable length buffer

steembase.types.**varint**(*n*)
Varint encoding

steembase.types.**varintdecode**(*data*)
Varint decoding

---

## Exceptions

steembase.exceptions.**decodeRPCErrorMsg**(*e*)
Helper function to decode the raised Exception and give it a python Exception class

CHAPTER 4

# Other

- genindex

# Python Module Index

## S

# Index

## F



## G

## H



## I



## J



## K



## L

## V



## W