

---

# **beem Documentation**

***Release 0.20.2***

**Holger Nahrstaedt**

**Oct 06, 2018**



---

## Contents

---

<b>1</b>	<b>About this Library</b>	<b>3</b>
<b>2</b>	<b>Quickstart</b>	<b>5</b>
<b>3</b>	<b>General</b>	<b>7</b>
<b>4</b>	<b>Indices and tables</b>	<b>169</b>
	<b>Python Module Index</b>	<b>171</b>



Steem is a blockchain-based rewards platform for publishers to monetize content and grow community.

It is based on *Graphene* (tm), a blockchain technology stack (i.e. software) that allows for fast transactions and ascalable blockchain solution. In case of Steem, it comes with decentralized publishing of content.

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



# CHAPTER 1

---

## About this Library

---

The purpose of *beem* is to simplify development of products and services that use the Steem blockchain. It comes with

- it's own (bip32-encrypted) wallet
- RPC interface for the Blockchain backend
- JSON-based blockchain objects (accounts, blocks, prices, markets, etc)
- a simple to use yet powerful API
- transaction construction and signing
- push notification API
- *and more*





## CHAPTER 2

---

### Quickstart

---

---

#### Note:

All methods that construct and sign a transaction can be given the `account=` parameter to identify the user that is going to be affected by this transaction, e.g.:

- the source account in a transfer
- the account that buys/sells an asset in the exchange
- the account whose collateral will be modified

**Important,** If no `account` is given, then the `default_account` according to the settings in `config` is used instead.

---

```
from beem import Steem
steem = Steem()
steem.wallet.unlock("wallet-passphrase")
account = Account("test", steem_instance=steem)
account.transfer("<to>", "<amount>", "<asset>", "<memo>")
```

```
from beem.blockchain import Blockchain
blockchain = Blockchain()
for op in blockchain.stream():
    print(op)
```

```
from beem.block import Block
print(Block(1))
```

```
from beem.account import Account
account = Account("test")
print(account.balances)
for h in account.history():
    print(h)
```

```
from beem.steem import Steem
stm = Steem()
stm.wallet.wipe(True)
stm.wallet.create("wallet-passphrase")
stm.wallet.unlock("wallet-passphrase")
stm.wallet.addPrivateKey("512345678")
stm.wallet.lock()
```

```
from beem.market import Market
market = Market("SBD:STEEM")
print(market.ticker())
market.steem.wallet.unlock("wallet-passphrase")
print(market.sell(300, 100) # sell 100 STEEM for 300 STEEM/SBD
```

### 3.1 Installation

The minimal working python version is 2.7.x. or 3.4.x

beem can be installed parallel to python-steem.

For Debian and Ubuntu, please ensure that the following packages are installed:

```
sudo apt-get install build-essential libssl-dev python-dev
```

For Fedora and RHEL-derivatives, please ensure that the following packages are installed:

```
sudo yum install gcc openssl-devel python-devel
```

For OSX, please do the following:

```
brew install openssl  
export CFLAGS="-I$(brew --prefix openssl)/include $CFLAGS"  
export LDFLAGS="-L$(brew --prefix openssl)/lib $LDFLAGS"
```

For Termux on Android, please install the following packages:

```
pkg install clang openssl-dev python-dev
```

Signing and Verify can be fasten (200 %) by installing cryptography:

```
pip install -U cryptography
```

Install beem by pip:

```
pip install -U beem
```

Sometimes this does not work. Please try:

```
pip3 install -U beem
```

or:

```
python -m pip install beem
```

### 3.1.1 Manual installation

You can install beem from this repository if you want the latest but possibly non-compiling version:

```
git clone https://github.com/holgern/beem.git
cd beem
python setup.py build
python setup.py install --user
```

Run tests after install:

```
pytest
```

### 3.1.2 Installing beem with conda-forge

Installing beem from the conda-forge channel can be achieved by adding conda-forge to your channels with:

```
conda config --add channels conda-forge
```

Once the conda-forge channel has been enabled, beem can be installed with:

```
conda install beem
```

Signing and Verify can be fasten (200 %) by installing cryptography:

```
conda install cryptography
```

### 3.1.3 Enable Logging

Add the following for enabling logging in your python script:

```
import logging
log = logging.getLogger(__name__)
logging.basicConfig(level=logging.INFO)
```

When you want to see only critical errors, replace the last line by:

```
logging.basicConfig(level=logging.CRITICAL)
```

## 3.2 Quickstart

### 3.2.1 Steem

The steem object is the connection to the Steem blockchain. By creating this object different options can be set.

---

**Note:** All init methods of beem classes can be given the `steem_instance=` parameter to assure that all objects use the same steem object. When the `steem_instance=` parameter is not used, the steem object is taken from `get_shared_steem_instance()`.

`get_shared_steem_instance()` returns a global instance of steem. It can be set by `set_shared_steem_instance` otherwise it is created on the first call.

---

```
from beem import Steem
from beem.account import Account
stm = Steem()
account = Account("test", steem_instance=stm)
```

```
from beem import Steem
from beem.account import Account
from beem.instance import set_shared_steem_instance
stm = Steem()
set_shared_steem_instance(stm)
account = Account("test")
```

### 3.2.2 Wallet and Keys

Each account has the following keys:

- Posting key (allows accounts to post, vote, edit, resteem and follow/mute)
- Active key (allows accounts to transfer, power up/down, voting for witness, ...)
- Memo key (Can be used to encrypt/decrypt memos)
- Owner key (The most important key, should not be used with beem)

Outgoing operation, which will be stored in the steem blockchain, have to be signed by a private key. E.g. Comment or Vote operation need to be signed by the posting key of the author or upvoter. Private keys can be provided to beem temporary or can be stored encrypted in a sql-database (wallet).

---

**Note:** Before using the wallet the first time, it has to be created and a password has to set. The wallet content is available to beem.py and all python scripts, which have access to the sql database file.

---

#### Creating a wallet

`steem.wallet.wipe(True)` is only necessary when there was already an wallet created.

```
from beem import Steem
steem = Steem()
steem.wallet.wipe(True)
steem.wallet.unlock("wallet-passphrase")
```

## Adding keys to the wallet

```
from beem import Steem
steem = Steem()
steem.wallet.unlock("wallet-passphrase")
steem.wallet.addPrivateKey("xxxxxxx")
steem.wallet.addPrivateKey("xxxxxxx")
```

## Using the keys in the wallet

```
from beem import Steem
steem = Steem()
steem.wallet.unlock("wallet-passphrase")
account = Account("test", steem_instance=steem)
account.transfer("<to>", "<amount>", "<asset>", "<memo>")
```

## Private keys can also set temporary

```
from beem import Steem
steem = Steem(keys=["xxxxxxxxx"])
account = Account("test", steem_instance=steem)
account.transfer("<to>", "<amount>", "<asset>", "<memo>")
```

## 3.2.3 Receiving information about blocks, accounts, votes, comments, market and witness

### Receive all Blocks from the Blockchain

```
from beem.blockchain import Blockchain
blockchain = Blockchain()
for op in blockchain.stream():
    print(op)
```

### Access one Block

```
from beem.block import Block
print(Block(1))
```

### Access an account

```
from beem.account import Account
account = Account("test")
print(account.balances)
for h in account.history():
    print(h)
```

### A single vote

```
from beem.vote import Vote
vote = Vote(u"@gtg/ffdhu-gtg-witness-log|gandalf")
print(vote.json())
```

All votes from an account

```
from beem.vote import AccountVotes
allVotes = AccountVotes("gtg")
```

Access a post

```
from beem.comment import Comment
comment = Comment("@gtg/ffdhu-gtg-witness-log")
print(comment["active_votes"])
```

Access the market

```
from beem.market import Market
market = Market("SBD:STEEM")
print(market.ticker())
```

Access a witness

```
from beem.witness import Witness
witness = Witness("gtg")
print(witness.is_active)
```

## 3.2.4 Sending transaction to the blockchain

Sending a Transfer

```
from beem import Steem
steem = Steem()
steem.wallet.unlock("wallet-passphrase")
account = Account("test", steem_instance=steem)
account.transfer("null", 1, "SBD", "test")
```

Upvote a post

```
from beem.comment import Comment
from beem import Steem
steem = Steem()
steem.wallet.unlock("wallet-passphrase")
comment = Comment("@gtg/ffdhu-gtg-witness-log", steem_instance=steem)
comment.upvote(weight=10, voter="test")
```

Publish a post to the blockchain

```
from beem import Steem
steem = Steem()
steem.wallet.unlock("wallet-passphrase")
steem.post("title", "body", author="test", tags=["a", "b", "c", "d", "e"], self_
↳ vote=True)
```

Sell STEEM on the market

```
from beem.market import Market
from beem import Steem
steem.wallet.unlock("wallet-passphrase")
market = Market("SBD:STEEM", steem_instance=steem)
```

(continues on next page)

(continued from previous page)

```
print(market.ticker())
market.steem.wallet.unlock("wallet-passphrase")
print(market.sell(300, 100)) # sell 100 STEEM for 300 STEEM/SBD
```

## 3.3 Tutorials

### 3.3.1 Bundle Many Operations

With Steem, you can bundle multiple operations into a single transactions. This can be used to do a multi-send (one sender, multiple receivers), but it also allows to use any other kind of operation. The advantage here is that the user can be sure that the operations are executed in the same order as they are added to the transaction.

A block can only include one vote operation and one comment operation from each sender.

```
from pprint import pprint
from beem import Steem
from beem.account import Account
from beem.comment import Comment
from beem.instance import set_shared_steem_instance

# not a real working key
wif = "5KQwrPbwdL6PhXujxW37FSSQZ1JiwsST4cqQzDeyXtP79zkvFD3"

stm = Steem(
    bundle=True, # Enable bundle broadcast
    # nobroadcast=True, # Enable this for testing
    keys=[wif],
)
# Set stm as shared instance
set_shared_steem_instance(stm)

# Account and Comment will use now stm
account = Account("test")

# Post
c = Comment("@gtg/witness-gtg-log")

account.transfer("test1", 1, "STEEM")
account.transfer("test2", 1, "STEEM")
account.transfer("test3", 1, "SBD")
# Upvote post with 25%
c.upvote(25, voter=account)

pprint(stm.broadcast())
```

### 3.3.2 Use nobroadcast for testing

When using *nobroadcast=True* the transaction is not broadcasted but printed.

```
from pprint import pprint
from beem import Steem
from beem.account import Account
```

(continues on next page)



(continued from previous page)

```

from beem.instance import set_shared_steem_instance

# Only for testing not a real working key
wif = "5KQwrPbwdL6PhXujxW37FSSQZ1JiwsST4cqQzDeyXtP79zkvFD3"

# set nobroadcast always to True, when testing
testnet = Steem(
    nobroadcast=True, # Set to false when want to go live
    keys=[wif],
)
# Set testnet as shared instance
set_shared_steem_instance(testnet)

# Account will use now testnet
account = Account("test")

pprint(account.transfer("test1", 1, "STEEM"))

```

When executing the script above, the output will be similar to the following:

```

Not broadcasting anything!
{'expiration': '2018-05-01T16:16:57',
 'extensions': [],
 'operations': [['transfer',
                   {'amount': '1.000 STEEM',
                    'from': 'test',
                    'memo': '',
                    'to': 'test1'}]],
 'ref_block_num': 33020,
 'ref_block_prefix': 2523628005,
 'signatures': [
  ↳ '1f57da50f241e70c229ed67b5d61898e792175c0f18ae29df8af414c46ae91eb5729c867b5d7dcc578368e7024e414c23'
  ↳ '']]

```

### 3.3.3 Clear BlockchainObject Caching

Each BlockchainObject (Account, Comment, Vote, Witness, Amount, ...) has a glocal cache. This cache stores all objects and could lead to increased memory consumption. The global cache can be cleared with a `clear_cache()` call from any BlockchainObject.

```

from pprint import pprint
from beem.account import Account

account = Account("test")
pprint(str(account._cache))
account1 = Account("test1")
pprint(str(account._cache))
pprint(str(account1._cache))
account.clear_cache()
pprint(str(account._cache))
pprint(str(account1._cache))

```

### 3.3.4 Simple Sell Script

```
from beem import Steem
from beem.market import Market
from beem.price import Price
from beem.amount import Amount

# Only for testing not a real working key
wif = "5KQwrPbwdL6PhXujxW37FSSQZlJiwsST4cqQzDeyXtP79zkvFD3"

#
# Instantiate Steem (pick network via API node)
#
steem = Steem(
    nobroadcast=True,    # <--- set this to False when you want to fire!
    keys=[wif]           # <--- use your real keys, when going live!
)

#
# This defines the market we are looking at.
# The first asset in the first argument is the *quote*
# Sell and buy calls always refer to the *quote*
#
market = Market("SBD:STEEM",
    steem_instance=steem
)

#
# Sell an asset for a price with amount (quote)
#
print(market.sell(
    Price(100.0, "STEEM/SBD"),
    Amount("0.01 SBD")
))
```

### 3.3.5 Sell at a timely rate

```
import threading
from beem import Steem
from beem.market import Market
from beem.price import Price
from beem.amount import Amount

# Only for testing not a real working key
wif = "5KQwrPbwdL6PhXujxW37FSSQZlJiwsST4cqQzDeyXtP79zkvFD3"

def sell():
    """ Sell an asset for a price with amount (quote)
    """
    print(market.sell(
        Price(100.0, "SBD/STEEM"),
        Amount("0.01 STEEM")
    ))

    threading.Timer(60, sell).start()
```

(continues on next page)

(continued from previous page)

```

if __name__ == "__main__":
    #
    # Instantiate Steem (pick network via API node)
    #
    steem = Steem(
        nobroadcast=True,    # <--- set this to False when you want to fire!
        keys=[wif]           # <--- use your real keys, when going live!
    )

    #
    # This defines the market we are looking at.
    # The first asset in the first argument is the *quote*
    # Sell and buy calls always refer to the *quote*
    #
    market = Market("STEEM:SBD",
        steem_instance=steem
    )

    sell()

```

### 3.3.6 Batch api calls on AppBase

Batch api calls are possible with AppBase RPC nodes. If you call a Api-Call with `add_to_queue=True` it is not submitted but stored in `rpc_queue`. When a call with `add_to_queue=False` (default setting) is started, the complete queue is sended at once to the node. The result is a list with replies.

```

from beem import Steem
stm = Steem("https://api.steemit.com")
stm.rpc.get_config(add_to_queue=True)
stm.rpc.rpc_queue

```

```
[{'method': 'condenser_api.get_config', 'jsonrpc': '2.0', 'params': [], 'id': 6}]
```

```

result = stm.rpc.get_block({"block_num":1}, api="block", add_to_queue=False)
len(result)

```

```
2
```

### 3.3.7 Account history

Lets calculate the curation reward from the last 7 days:

```

from datetime import datetime, timedelta
from beem.account import Account
from beem.amount import Amount

acc = Account("gtg")
stop = datetime.utcnow() - timedelta(days=7)
reward_vests = Amount("0 VESTS")
for reward in acc.history_reverse(stop=stop, only_ops=["curation_reward"]):

```

(continues on next page)

(continued from previous page)

```

        reward_vests += Amount(reward['reward'])
    curation_rewards_SP = acc.steem.vests_to_sp(reward_vests.amount)
    print("Rewards are %.3f SP" % curation_rewards_SP)

```

Lets display all Posts from an account:

```

from beem.account import Account
from beem.comment import Comment
from beem.exceptions import ContentDoesNotExistsException
account = Account("holger80")
c_list = {}
for c in map(Comment, account.history(only_ops=["comment"])):
    if c.permlink in c_list:
        continue
    try:
        c.refresh()
    except ContentDoesNotExistsException:
        continue
    c_list[c.permlink] = 1
    if not c.is_comment():
        print("%s " % c.title)

```

### 3.3.8 Transactionbuilder

Sign transactions with beem without using the wallet and build the transaction by hand. Example with one operation with and without the wallet:

```

from beem import Steem
from beem.transactionbuilder import TransactionBuilder
from beembase import operations
stm = Steem()
# Uncomment the following when using a wallet:
# stm.wallet.unlock("secret_password")
tx = TransactionBuilder(stem_instance=stm)
op = operations.Transfer(**{"from": 'user_a',
                             "to": 'user_b',
                             "amount": '1.000 SBD',
                             "memo": 'test 2'})

tx.appendOps(op)
# Comment appendWif out and uncomment appendSigner when using a stored key from the_
↪wallet
tx.appendWif('5.....') # `user_a`
# tx.appendSigner('user_a', 'active')
tx.sign()
tx.broadcast()

```

Example with signing and broadcasting two operations:

```

from beem import Steem
from beem.transactionbuilder import TransactionBuilder
from beembase import operations
stm = Steem()
# Uncomment the following when using a wallet:
# stm.wallet.unlock("secret_password")
tx = TransactionBuilder(stem_instance=stm)

```

(continues on next page)

(continued from previous page)

```
ops = []
op = operations.Transfer(**{"from": 'user_a',
                           "to": 'user_b',
                           "amount": '1.000 SBD',
                           "memo": 'test 2'})

ops.append(op)
op = operations.Vote(**{"voter": v,
                       "author": author,
                       "permlink": permlink,
                       "weight": int(percent * 100)})

ops.append(op)
tx.appendOps(ops)
# Comment appendWif out and uncomment appendSigner when using a stored key from the
↪ wallet
tx.appendWif('5.....') # `user_a`
# tx.appendSigner('user_a', 'active')
tx.sign()
tx.broadcast()
```

## 3.4 beempy CLI

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

### 3.4.1 Using the Wallet

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

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

You can change the password via *changewalletpassphrase* command.

```
beempy changewalletpassphrase
```

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

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

```
UNLOCK=mysecretpassword beempy transfer <recipient_name> 100 STEEM
```

### 3.4.2 Common Commands

First, you may like to import your Steem account:

```
beempy importaccount
```

You can also import individual private keys:

```
beempy addkey <private_key>
```

Listing accounts:

```
beempy listaccounts
```

Show balances:

```
beempy balance account_name1 account_name2
```

Sending funds:

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

Upvoting a post:

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

### 3.4.3 Setting Defaults

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

```
beempy set default_account test  
beempy set default_vote_weight 100  
  
beempy config  
+-----+-----+  
| Key           | Value |  
+-----+-----+  
| default_account | test  |  
| default_vote_weight | 100  |  
+-----+-----+
```

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

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

### 3.4.4 Commands

#### **beempy**

```
beempy [OPTIONS] COMMAND1 [ARGS]... [COMMAND2 [ARGS]...]...
```

#### **Options**

- n, --node <node>**  
URL for public Steem API (e.g. <https://api.steemit.com>)
- o, --offline**  
Prevent connecting to network
- d, --no-broadcast**  
Do not broadcast

**-p, --no-wallet**  
Do not load the wallet

**-x, --unsigned**  
Nothing will be signed

**-l, --create-link**  
Creates steemconnect links from all broadcast operations

**-s, --steemconnect**  
Uses a steemconnect token to broadcast (only broadcast operation with posting permission)

**-e, --expires <expires>**  
Delay in seconds until transactions are supposed to expire(defaults to 60)

**-v, --verbose <verbose>**  
Verbosity

**--version**  
Show the version and exit.

## addkey

Add key to wallet

When no [OPTION] is given, a password prompt for unlocking the wallet and a prompt for entering the private key are shown.

```
beempy addkey [OPTIONS]
```

## Options

**--unsafe-import-key <unsafe\_import\_key>**  
Private key to import to wallet (unsafe, unless shell history is deleted afterwards)

## addtoken

Add key to wallet

When no [OPTION] is given, a password prompt for unlocking the wallet and a prompt for entering the private key are shown.

```
beempy addtoken [OPTIONS] NAME
```

## Options

**--unsafe-import-token <unsafe\_import\_token>**  
Private key to import to wallet (unsafe, unless shell history is deleted afterwards)

## Arguments

**NAME**  
Required argument

## allow

Allow an account/key to interact with your account

**foreign\_account:** The account or key that will be allowed to interact with account. When not given, password will be asked, from which a public key is derived. This derived key will then interact with your account.

```
beempy allow [OPTIONS] [FOREIGN_ACCOUNT]
```

### Options

**--permission** <permission>

The permission to grant (defaults to “posting”)

**-a, --account** <account>

The account to allow action for

**--weight** <weight>

The weight to use instead of the (full) threshold. If the weight is smaller than the threshold, additional signatures are required

**--threshold** <threshold>

The permission’s threshold that needs to be reached by signatures to be able to interact

### Arguments

**FOREIGN\_ACCOUNT**

Optional argument

## approvewitness

Approve a witnesses

```
beempy approvewitness [OPTIONS] WITNESS
```

### Options

**-a, --account** <account>

Your account

### Arguments

**WITNESS**

Required argument

## balance

Shows balance



```
beempy balance [OPTIONS] [ACCOUNT]...
```

## Arguments

### ACCOUNT

Optional argument(s)

## broadcast

broadcast a signed transaction

```
beempy broadcast [OPTIONS]
```

## Options

**--file** <file>

Load transaction from file. If “-“, read from stdin (defaults to “-“)

## buy

Buy STEEM or SBD from the internal market

Limit buy price denoted in (SBD per STEEM)

```
beempy buy [OPTIONS] AMOUNT ASSET [PRICE]
```

## Options

**-a, --account** <account>

Buy with this account (defaults to “default\_account”)

**--orderid** <orderid>

Set an orderid

## Arguments

### AMOUNT

Required argument

### ASSET

Required argument

### PRICE

Optional argument

## cancel

Cancel order in the internal market

```
beempy cancel [OPTIONS] ORDERID
```

## Options

**-a, --account** <account>  
Sell with this account (defaults to “default\_account”)

## Arguments

**ORDERID**  
Required argument

## changewalletpassphrase

Change wallet password

```
beempy changewalletpassphrase [OPTIONS]
```

## claimreward

Claim reward balances

By default, this will claim all outstanding balances.

```
beempy claimreward [OPTIONS] [ACCOUNT]
```

## Options

**--reward\_steem** <reward\_steem>  
Amount of STEEM you would like to claim

**--reward\_sbd** <reward\_sbd>  
Amount of SBD you would like to claim

**--reward\_vests** <reward\_vests>  
Amount of VESTS you would like to claim

**--claim\_all\_steem**  
Claim all STEEM, overwrites reward\_steem

**--claim\_all\_sbd**  
Claim all SBD, overwrites reward\_sbd

**--claim\_all\_vests**  
Claim all VESTS, overwrites reward\_vests

## Arguments

### ACCOUNT

Optional argument

## config

Shows local configuration

```
beempy config [OPTIONS]
```

## convert

Convert STEEMDollars to Steem (takes a week to settle)

```
beempy convert [OPTIONS] AMOUNT
```

## Options

**-a, --account** <account>  
Powerup from this account

## Arguments

### AMOUNT

Required argument

## createwallet

Create new wallet with a new password

```
beempy createwallet [OPTIONS]
```

## Options

**--wipe**  
Wipe old wallet without prompt.

## curation

Lists curation rewards of all votes for authorperm

When authorperm is empty or “all”, the curation rewards for all account votes are shown.

authorperm can also be a number. e.g. 5 is equivalent to the fifth account vote in the given time duration (default is 7 days)

```
beempy curation [OPTIONS] [AUTHORPERM]
```

## Options

- a, --account** <account>  
Show only curation for this account
- m, --limit** <limit>  
Show only the first minutes
- v, --min-vote** <min\_vote>  
Show only votes higher than the given value
- w, --max-vote** <max\_vote>  
Show only votes lower than the given value
- x, --min-performance** <min\_performance>  
Show only votes with performance higher than the given value
- y, --max-performance** <max\_performance>  
Show only votes with performance lower than the given value
- payout** <payout>  
Show the curation for a potential payout in SBD as float
- e, --export** <export>  
Export results to HTML-file
- s, --short**  
Show only Curation without sum
- l, --length** <length>  
Limits the permalink character length
- p, --permalink**  
Show the permalink for each entry
- t, --title**  
Show the title for each entry
- d, --days** <days>  
Limit shown rewards by this amount of days (default: 7), max is 7 days.

## Arguments

**AUTHORPERM**  
Optional argument

## currentnode

Sets the currently working node at the first place in the list

```
beempy currentnode [OPTIONS]
```

## Options

### **--version**

Returns only the raw version value

### **--url**

Returns only the raw url value

## delegate

Delegate (start delegate VESTS to another account)

amount is in VESTS / Steem

```
beempy delegate [OPTIONS] AMOUNT TO_ACCOUNT
```

## Options

### **-a, --account <account>**

Powerup from this account

## Arguments

### **AMOUNT**

Required argument

### **TO\_ACCOUNT**

Required argument

## delkey

Delete key from the wallet

PUB is the public key from the private key which will be deleted from the wallet

```
beempy delkey [OPTIONS] PUB
```

## Options

### **--confirm**

Please confirm!

## Arguments

### **PUB**

Required argument

## delprofile

Delete a variable in an account's profile

```
beempy delprofile [OPTIONS] VARIABLE...
```

### Options

**-a, --account** <account>  
delprofile as this user

### Arguments

**VARIABLE**  
Required argument(s)

## deltoken

Delete name from the wallet

name is the public name from the private token which will be deleted from the wallet

```
beempy deltoken [OPTIONS] NAME
```

### Options

**--confirm**  
Please confirm!

### Arguments

**NAME**  
Required argument

## disallow

Remove allowance an account/key to interact with your account

```
beempy disallow [OPTIONS] [FOREIGN_ACCOUNT]
```

### Options

**--permission** <permission>  
The permission to grant (defaults to "posting")

**-a, --account** <account>  
The account to disallow action for

**--threshold** <threshold>

The permission's threshold that needs to be reached by signatures to be able to interact

## Arguments

**FOREIGN\_ACCOUNT**

Optional argument

## disapprovewitness

Disapprove a witnesses

```
beempy disapprovewitness [OPTIONS] WITNESS
```

## Options

**-a, --account** <account>

Your account

## Arguments

**WITNESS**

Required argument

## downvote

Downvote a post/comment

POST is @author/permlink

```
beempy downvote [OPTIONS] POST [VOTE_WEIGHT]
```

## Options

**-a, --account** <account>

Voter account name

**-w, --weight** <weight>

Vote weight (from 0.1 to 100.0)

## Arguments

**POST**

Required argument

**VOTE\_WEIGHT**

Optional argument

## featureflags

Get the account's feature flags.

The request has to be signed by the requested account or an admin account.

```
beempy featureflags [OPTIONS] [ACCOUNT]
```

### Options

**-s, --signing-account** <signing\_account>  
Signing account, when empty account is used.

### Arguments

**ACCOUNT**  
Optional argument

## follow

Follow another account

```
beempy follow [OPTIONS] FOLLOW
```

### Options

**-a, --account** <account>  
Follow from this account

**--what** <what>  
Follow these objects (defaults to ["blog"])

### Arguments

**FOLLOW**  
Required argument

## follower

Get information about followers

```
beempy follower [OPTIONS] [ACCOUNT]...
```

### Arguments

**ACCOUNT**  
Optional argument(s)



## following

Get information about following

```
beempy following [OPTIONS] [ACCOUNT]...
```

## Arguments

### ACCOUNT

Optional argument(s)

## importaccount

Import an account using a passphrase

```
beempy importaccount [OPTIONS] ACCOUNT
```

## Options

**--roles** <roles>

Import specified keys (owner, active, posting, memo).

## Arguments

### ACCOUNT

Required argument

## info

Show basic blockchain info

General information about the blockchain, a block, an account, a post/comment and a public key

```
beempy info [OPTIONS] [OBJECTS]...
```

## Arguments

### OBJECTS

Optional argument(s)

## interest

Get information about interest payment

```
beempy interest [OPTIONS] [ACCOUNT]...
```

## Arguments

### ACCOUNT

Optional argument(s)

## keygen

Creates a new random brain key and prints its derived private key and public key. The generated key is not stored.

```
beempy keygen [OPTIONS]
```

## Options

### --import-brain-key

Imports a brain key and derives a private and public key

### --sequence <sequence>

Sequence number, influences the derived private key. (default is 0)

## listaccounts

Show stored accounts

```
beempy listaccounts [OPTIONS]
```

## listkeys

Show stored keys

```
beempy listkeys [OPTIONS]
```

## listtoken

Show stored token

```
beempy listtoken [OPTIONS]
```

## mute

Mute another account

```
beempy mute [OPTIONS] MUTE
```

## Options

- a, --account** <account>  
Mute from this account
- what** <what>  
Mute these objects (defaults to ["ignore"])

## Arguments

- MUTE**  
Required argument

### muter

Get information about muter

```
beempy muter [OPTIONS] [ACCOUNT]...
```

## Arguments

- ACCOUNT**  
Optional argument(s)

### muting

Get information about muting

```
beempy muting [OPTIONS] [ACCOUNT]...
```

## Arguments

- ACCOUNT**  
Optional argument(s)

### newaccount

Create a new account

```
beempy newaccount [OPTIONS] ACCOUNTNAME
```

## Options

- a, --account** <account>  
Account that pays the fee

**--fee** <fee>

Base Fee to pay. Delegate the rest.

## Arguments

**ACCOUNTNAME**

Required argument

## nextnode

Uses the next node in list

```
beempy nextnode [OPTIONS]
```

## Options

**--results**

Shows result of changing the node.

## openorders

Show open orders

```
beempy openorders [OPTIONS] [ACCOUNT]
```

## Arguments

**ACCOUNT**

Optional argument

## orderbook

Obtain orderbook of the internal market

```
beempy orderbook [OPTIONS]
```

## Options

**--chart**

Enable charting

**-l, --limit** <limit>

Limit number of returned open orders (default 25)

**--show-date**

Show dates

**-w, --width** <width>  
Plot width (default 75)

**-h, --height** <height>  
Plot height (default 15)

**--ascii**  
Use only ascii symbols

### parsewif

Parse a WIF private key without importing

```
beempy parsewif [OPTIONS]
```

### Options

**--unsafe-import-key** <unsafe\_import\_key>  
WIF key to parse (unsafe, unless shell history is deleted afterwards)

### pending

Lists pending rewards

```
beempy pending [OPTIONS] [ACCOUNTS]...
```

### Options

**-s, --only-sum**  
Show only the sum

**-p, --post**  
Show pending post payout

**-c, --comment**  
Show pending comments payout

**-v, --curation**  
Shows pending curation

**-l, --length** <length>  
Limits the permalink character length

**-a, --author**  
Show the author for each entry

**-e, --permalink**  
Show the permalink for each entry

**-t, --title**  
Show the title for each entry

**-d, --days** <days>  
Limit shown rewards by this amount of days (default: 7), max is 7 days.

## Arguments

### ACCOUNTS

Optional argument(s)

## permissions

Show permissions of an account

```
beempy permissions [OPTIONS] [ACCOUNT]
```

## Arguments

### ACCOUNT

Optional argument

## pingnode

Returns the answer time in milliseconds

```
beempy pingnode [OPTIONS]
```

## Options

### --raw

Returns only the raw value

### --sort

Sort all nodes by ping value

### --remove

Remove node with errors from list

### --threading

Use a thread for each node

## power

Shows vote power and bandwidth

```
beempy power [OPTIONS] [ACCOUNT]...
```

## Arguments

### ACCOUNT

Optional argument(s)

## powerdown

Power down (start withdrawing VESTS from Steem POWER)

amount is in VESTS

```
beempy powerdown [OPTIONS] AMOUNT
```

### Options

**-a, --account** <account>  
Powerup from this account

### Arguments

**AMOUNT**  
Required argument

## powerdownroute

Setup a powerdown route

```
beempy powerdownroute [OPTIONS] TO
```

### Options

**--percentage** <percentage>  
The percent of the withdraw to go to the “to” account

**-a, --account** <account>  
Powerup from this account

**--auto\_vest**  
Set to true if the from account should receive the VESTS as VESTS, or false if it should receive them as STEEM.

### Arguments

**TO**  
Required argument

## powerup

Power up (vest STEEM as STEEM POWER)

```
beempy powerup [OPTIONS] AMOUNT
```

## Options

**-a, --account** <account>  
Powerup from this account

**--to** <to>  
Powerup this account

## Arguments

**AMOUNT**  
Required argument

## pricehistory

Show price history

```
beempy pricehistory [OPTIONS]
```

## Options

**-w, --width** <width>  
Plot width (default 75)

**-h, --height** <height>  
Plot height (default 15)

**--ascii**  
Use only ascii symbols

## resteem

Resteem an existing post

```
beempy resteem [OPTIONS] IDENTIFIER
```

## Options

**-a, --account** <account>  
Resteem as this user

## Arguments

**IDENTIFIER**  
Required argument



## rewards

Lists received rewards

```
beempy rewards [OPTIONS] [ACCOUNTS]...
```

### Options

- s, --only-sum**  
Show only the sum
- p, --post**  
Show post payout
- c, --comment**  
Show comments payout
- v, --curation**  
Shows curation
- l, --length <length>**  
Limits the permalink character length
- a, --author**  
Show the author for each entry
- e, --permalink**  
Show the permalink for each entry
- t, --title**  
Show the title for each entry
- d, --days <days>**  
Limit shown rewards by this amount of days (default: 7)

### Arguments

**ACCOUNTS**  
Optional argument(s)

## sell

Sell STEEM or SBD from the internal market

Limit sell price denoted in (SBD per STEEM)

```
beempy sell [OPTIONS] AMOUNT ASSET [PRICE]
```

### Options

- a, --account <account>**  
Sell with this account (defaults to “default\_account”)
- orderid <orderid>**  
Set an orderid

## Arguments

### AMOUNT

Required argument

### ASSET

Required argument

### PRICE

Optional argument

## set

Set default\_account, default\_vote\_weight or nodes

set [key] [value]

Examples:

Set the default vote weight to 50 %: set default\_vote\_weight 50

```
beempy set [OPTIONS] KEY VALUE
```

## Arguments

### KEY

Required argument

### VALUE

Required argument

## setprofile

Set a variable in an account's profile

```
beempy setprofile [OPTIONS] [VARIABLE] [VALUE]
```

## Options

**-a, --account** <account>  
setprofile as this user

**-p, --pair** <pair>  
"Key=Value" pairs

## Arguments

### VARIABLE

Optional argument

### VALUE

Optional argument

## sign

Sign a provided transaction with available and required keys

```
beempy sign [OPTIONS]
```

### Options

**--file** <file>

Load transaction from file. If “-“, read from stdin (defaults to “-“)

## ticker

Show ticker

```
beempy ticker [OPTIONS]
```

### Options

**-i, --sbd-to-steem**

Show ticker in SBD/STEEM

## tradehistory

Show price history

```
beempy tradehistory [OPTIONS]
```

### Options

**-d, --days** <days>

Limit the days of shown trade history (default 7)

**--hours** <hours>

Limit the intervall history intervall (default 2 hours)

**-i, --sbd-to-steem**

Show ticker in SBD/STEEM

**-l, --limit** <limit>

Limit number of trades which is fetched at each intervall point (default 100)

**-w, --width** <width>

Plot width (default 75)

**-h, --height** <height>

Plot height (default 15)

**--ascii**

Use only ascii symbols

## transfer

Transfer SBD/STEEM

```
beempy transfer [OPTIONS] TO AMOUNT ASSET [MEMO]
```

### Options

**-a, --account** <account>  
Transfer from this account

### Arguments

**TO**  
Required argument

**AMOUNT**  
Required argument

**ASSET**  
Required argument

**MEMO**  
Optional argument

## unfollow

Unfollow/Unmute another account

```
beempy unfollow [OPTIONS] UNFOLLOW
```

### Options

**-a, --account** <account>  
UnFollow/UnMute from this account

### Arguments

**UNFOLLOW**  
Required argument

## updatememokey

Update an account's memo key

```
beempy updatememokey [OPTIONS]
```

## Options

- a, --account** <account>  
The account to update memo key action for
- key** <key>  
The new memo key

## updatenodes

Update the nodelist from @fullnodeupdate

```
beempy updatenodes [OPTIONS]
```

## Options

- s, --show**  
Prints the updated nodes
- t, --test**  
Do change the node list, only print the newest nodes setup.
- h, --only-https**  
Use only https nodes.
- w, --only-wss**  
Use only websocket nodes.
- a, --only-appbase**  
Use only appbase nodes
- n, --only-non-appbase**  
Use only non-appbase nodes

## upvote

Upvote a post/comment

POST is @author/permlink

```
beempy upvote [OPTIONS] POST [VOTE_WEIGHT]
```

## Options

- w, --weight** <weight>  
Vote weight (from 0.1 to 100.0)
- a, --account** <account>  
Voter account name

## Arguments

### POST

Required argument

### VOTE\_WEIGHT

Optional argument

## userdata

Get the account's email address and phone number.

The request has to be signed by the requested account or an admin account.

```
beempy userdata [OPTIONS] [ACCOUNT]
```

## Options

**-s, --signing-account** <signing\_account>  
Signing account, when empty account is used.

## Arguments

### ACCOUNT

Optional argument

## verify

Returns the public signing keys for a block

```
beempy verify [OPTIONS] [BLOCKNUMBER]
```

## Options

**-t, --trx** <trx>  
Show only one transaction number

**-u, --use-api**  
Uses the get\_potential\_signatures api call

## Arguments

### BLOCKNUMBER

Optional argument

## votes

List outgoing/incoming account votes

```
beempy votes [OPTIONS] [ACCOUNT]
```

### Options

- direction** <direction>  
in or out
- o, --outgoing**  
Show outgoing votes
- i, --incoming**  
Show incoming votes
- d, --days** <days>  
Limit shown vote history by this amount of days (default: 2)
- e, --export** <export>  
Export results to TXT-file

### Arguments

**ACCOUNT**  
Optional argument

## walletinfo

Show info about wallet

```
beempy walletinfo [OPTIONS]
```

### Options

- test-unlock**  
test if unlock is successful

## witness

List witness information

```
beempy witness [OPTIONS] WITNESS
```

### Arguments

**WITNESS**  
Required argument

## witnesscreate

Create a witness

```
beempy witnesscreate [OPTIONS] WITNESS PUB_SIGNING_KEY
```

### Options

**--maximum\_block\_size** <maximum\_block\_size>  
Max block size

**--account\_creation\_fee** <account\_creation\_fee>  
Account creation fee

**--sbd\_interest\_rate** <sbd\_interest\_rate>  
SBD interest rate in percent

**--url** <url>  
Witness URL

### Arguments

**WITNESS**  
Required argument

**PUB\_SIGNING\_KEY**  
Required argument

## witnessdisable

Disable a witness

```
beempy witnessdisable [OPTIONS] WITNESS
```

### Arguments

**WITNESS**  
Required argument

## witnessenable

Enable a witness

```
beempy witnessenable [OPTIONS] WITNESS SIGNING_KEY
```

### Arguments

**WITNESS**  
Required argument



**SIGNING\_KEY**

Required argument

**witnesses**

List witnesses

```
beempy witnesses [OPTIONS] [ACCOUNT]
```

**Options****--limit** <limit>

How many witnesses should be shown

**Arguments****ACCOUNT**

Optional argument

**witnessfeed**

Publish price feed for a witness

```
beempy witnessfeed [OPTIONS] WITNESS [WIF]
```

**Options****-b, --base** <base>

Set base manually, when not set the base is automatically calculated.

**-q, --quote** <quote>

Steem quote manually, when not set the base is automatically calculated.

**--support-peg**Supports peg adjusting the quote, is overwritten by `--set-quote!`**Arguments****WITNESS**

Required argument

**WIF**

Optional argument

## witnessproperties

Update witness properties of witness WITNESS with the witness signing key WIF

```
beempy witnessproperties [OPTIONS] WITNESS WIF
```

### Options

```
--account_creation_fee <account_creation_fee>
    Account creation fee (float)
--account_subsidy_budget <account_subsidy_budget>
    Account subsidy per block
--account_subsidy_decay <account_subsidy_decay>
    Per block decay of the account subsidy pool
--maximum_block_size <maximum_block_size>
    Max block size
--sbd_interest_rate <sbd_interest_rate>
    SBD interest rate in percent
--new_signing_key <new_signing_key>
    Set new signing key
--url <url>
    Witness URL
```

### Arguments

```
WITNESS
    Required argument
WIF
    Required argument
```

## witnessupdate

Change witness properties

```
beempy witnessupdate [OPTIONS]
```

### Options

```
--witness <witness>
    Witness name
--maximum_block_size <maximum_block_size>
    Max block size
--account_creation_fee <account_creation_fee>
    Account creation fee
```

```
--sbd_interest_rate <sbd_interest_rate>
    SBD interest rate in percent

--url <url>
    Witness URL

--signing_key <signing_key>
    Signing Key
```

### 3.4.5 beempy -help

You can see all available commands with `beempy --help`

```
~ % beempy --help
Usage: cli.py [OPTIONS] COMMAND1 [ARGS]... [COMMAND2 [ARGS]...]...

Options:
  -n, --node TEXT          URL for public Steem API (e.g.
                           https://api.steemit.com)
  -o, --offline            Prevent connecting to network
  -d, --no-broadcast       Do not broadcast
  -p, --no-wallet          Do not load the wallet
  -x, --unsigned           Nothing will be signed
  -e, --expires INTEGER    Delay in seconds until transactions are supposed to
                           expire(defaults to 60)
  -v, --verbose INTEGER    Verbosity
  --version                Show the version and exit.
  --help                  Show this message and exit.

Commands:
  addkey                  Add key to wallet When no [OPTION] is given,...
  allow                  Allow an account/key to interact with your...
  approvewitness         Approve a witnesses
  balance                Shows balance
  broadcast              broadcast a signed transaction
  buy                    Buy STEEM or SBD from the internal market...
  cancel                 Cancel order in the internal market
  changewalletpassphrase Change wallet password
  claimreward            Claim reward balances By default, this will...
  config                 Shows local configuration
  convert                Convert STEEMDollars to Steem (takes a week...
  createwallet           Create new wallet with a new password
  currentnode            Sets the currently working node at the first...
  delkey                 Delete key from the wallet PUB is the public...
  delprofile             Delete a variable in an account's profile
  disallow               Remove allowance an account/key to interact...
  disapprovewitness      Disapprove a witnesses
  downvote              Downvote a post/comment POST is...
  follow                Follow another account
  follower              Get information about followers
  following              Get information about following
  importaccount          Import an account using a passphrase
  info                  Show basic blockchain info General...
  interest              Get information about interest payment
  listaccounts           Show stored accounts
  listkeys              Show stored keys
  mute                  Mute another account
```

(continues on next page)

(continued from previous page)

muter	Get information about muter
muting	Get information about muting
newaccount	Create a new account
nextnode	Uses the <code>next</code> node <code>in</code> list
openorders	Show <code>open</code> orders
orderbook	Obtain orderbook of the internal market
parsewif	Parse a WIF private key without importing
permissions	Show permissions of an account
pingnode	Returns the answer time <code>in</code> milliseconds
power	Shows vote power <code>and</code> bandwidth
powerdown	Power down (start withdrawing VESTS from...
powerdownroute	Setup a powerdown route
powerup	Power up (vest STEEM <code>as</code> STEEM POWER)
pricehistory	Show price history
resteam	Resteem an existing post
sell	Sell STEEM <code>or</code> SBD <code>from the</code> internal market...
set	Set default_account, default_vote_weight <code>or</code> ...
setprofile	Set a variable <code>in</code> an account's <code>profile</code>
sign	Sign a provided transaction <code>with</code> available...
ticker	Show ticker
tradehistory	Show price history
transfer	Transfer SBD/STEEM
unfollow	Unfollow/Unmute another account
updatememokey	Update an account's <code>memo key</code>
upvote	Upvote a post/comment POST <code>is</code> ...
votes	List outgoing/incoming account votes
walletinfo	Show info about wallet
witnesscreate	Create a witness
witnesses	List witnesses
witnessupdate	Change witness properties

## 3.5 Configuration

The pysteem library comes with its own local configuration database that stores information like

- API node URLs
- default account name
- the encrypted master password
- the default voting weight
- if keyring should be used for unlocking the wallet

and potentially more.

You can access those variables like a regular dictionary by using

```
from beem import Steem
steem = Steem()
print(steem.config.items())
```

Keys can be added and changed like they are for regular dictionaries.

If you don't want to load the `beem.Steem` class, you can load the configuration directly by using:

```
from beem.storage import configStorage as config
```

It is also possible to access the configuration with the commandline tool *beempy*:

```
beempy config
```

### 3.5.1 API node URLs

The default node URLs which will be used when *node* is *None* in *beem.Steem* class is stored in *config["nodes"]* as string. The list can be get and set by:

```
from beem import Steem
steem = Steem()
node_list = steem.get_default_nodes()
node_list = node_list[1:] + [node_list[0]]
steem.set_default_nodes(node_list)
```

*beempy* can also be used to set nodes:

```
beempy set nodes wss://steemd.privex.io
beempy set nodes ["wss://steemd.privex.io", 'wss://gtg.steem.house:8090']"
```

The default nodes can be resetted to the default value. When the first node does not answer, *steem* should be set to the offline mode. This can be done by:

```
beempy -o set nodes ""
```

or

```
from beem import Steem
steem = Steem(offline=True)
steem.set_default_nodes("")
```

### 3.5.2 Default account

The default account name is used in some functions, when no account name is given. It is also used in *beempy* for all account related functions.

```
from beem import Steem
steem = Steem()
steem.set_default_account("test")
steem.config["default_account"] = "test"
```

or by *beempy* with

```
beempy set default_account test
```

### 3.5.3 Default voting weight

The default vote weight is used for voting, when no vote weight is given.

```
from beem import Steem
steem = Steem()
steem.config["default_vote_weight"] = 100
```

or by beempy with

```
beempy set default_vote_weight 100
```

### 3.5.4 Setting password\_storage

The password\_storage can be set to:

- environment, this is the default setting. The master password for the wallet can be provided in the environment variable *UNLOCK*.
- keyring (when set with beempy, it asks for the wallet password)

```
beempy set password_storage environment
beempy set password_storage keyring
```

#### Environment variable for storing the master password

When *password\_storage* is set to *environment*, the master password can be stored in *UNLOCK* for unlocking automatically the wallet.

#### Keyring support for beempy and wallet

In order to use keyring for storing the wallet password, the following steps are necessary:

- Install keyring: *pip install keyring*
- Change *password\_storage* to *keyring* with *beempy* and enter the wallet password.

It also possible to change the password in the keyring by

```
python -m keyring set beem wallet
```

The stored master password can be displayed in the terminal by

```
python -m keyring get beem wallet
```

When keyring is set as *password\_storage* and the stored password in the keyring is identically to the set master password of the wallet, the wallet is automatically unlocked everytime it is used.

#### Testing if unlocking works

Testing if the master password is correctly provided by keyring or the *UNLOCK* variable:

```
from beem import Steem
steem = Steem()
print(steem.wallet.locked())
```

When the output is False, automatic unlocking with keyring or the *UNLOCK* variable works. It can also tested by beempy with

```
beempy walletinfo --test-unlock
```

When no password prompt is shown, unlocking with keyring or the *UNLOCK* variable works.

## 3.6 Api Definitions

### 3.6.1 condenser\_api

#### **broadcast\_block**

not implemented

#### **broadcast\_transaction**

```
from beem.transactionbuilder import TransactionBuilder
t = TransactionBuilder()
t.broadcast()
```

#### **broadcast\_transaction\_synchronous**

```
from beem.transactionbuilder import TransactionBuilder
t = TransactionBuilder()
t.broadcast()
```

#### **get\_account\_bandwidth**

```
from beem.account import Account
account = Account("test")
account.get_account_bandwidth()
```

#### **get\_account\_count**

```
from beem.blockchain import Blockchain
b = Blockchain()
b.get_account_count()
```

#### **get\_account\_history**

```
from beem.account import Account
acc = Account("steemit")
for h in acc.get_account_history(1,0):
    print(h)
```

### get\_account\_reputations

```
from beem.blockchain import Blockchain
b = Blockchain()
for h in b.get_account_reputations():
    print(h)
```

### get\_account\_votes

```
from beem.account import Account
acc = Account("gtg")
for h in acc.get_account_votes():
    print(h)
```

### get\_active\_votes

```
from beem.vote import ActiveVotes
acc = Account("gtg")
post = acc.get_feed(0,1)[0]
a = ActiveVotes(post["authorperm"])
a.printAsTable()
```

### get\_active\_witnesses

```
from beem.witness import Witnesses
w = Witnesses()
w.printAsTable()
```

### get\_block

```
from beem.block import Block
print(Block(1))
```

### get\_block\_header

```
from beem.block import BlockHeader
print(BlockHeader(1))
```

### get\_blog

```
from beem.account import Account
acc = Account("gtg")
for h in acc.get_blog():
    print(h)
```



### get\_blog\_authors

```
from beem.account import Account
acc = Account("gtg")
for h in acc.get_blog_authors():
    print(h)
```

### get\_blog\_entries

```
from beem.account import Account
acc = Account("gtg")
for h in acc.get_blog_entries():
    print(h)
```

### get\_chain\_properties

```
from beem import Steem
stm = Steem()
print(stm.get_chain_properties())
```

### get\_comment\_discussions\_by\_payout

```
from beem.discussions import Query, Comment_discussions_by_payout
q = Query(limit=10)
for h in Comment_discussions_by_payout(q):
    print(h)
```

### get\_config

```
from beem import Steem
stm = Steem()
print(stm.get_config())
```

### get\_content

```
from beem.account import Account
from beem.comment import Comment
acc = Account("gtg")
post = acc.get_feed(0,1)[0]
print(Comment(post["authorperm"]))
```

### get\_content\_replies

```
from beem.account import Account
from beem.comment import Comment
acc = Account("gtg")
post = acc.get_feed(0,1)[0]
c = Comment(post["authorperm"])
for h in c.get_replies():
    print(h)
```

### get\_conversion\_requests

```
from beem.account import Account
acc = Account("gtg")
print(acc.get_conversion_requests())
```

### get\_current\_median\_history\_price

```
from beem import Steem
stm = Steem()
print(stm.get_current_median_history())
```

### get\_discussions\_by\_active

```
from beem.discussions import Query, Discussions_by_active
q = Query(limit=10)
for h in Discussions_by_active(q):
    print(h)
```

### get\_discussions\_by\_author\_before\_date

```
from beem.discussions import Query, Discussions_by_author_before_date
for h in Discussions_by_author_before_date(limit=10, author="gtg"):
    print(h)
```

### get\_discussions\_by\_blog

```
from beem.discussions import Query, Discussions_by_blog
q = Query(limit=10)
for h in Discussions_by_blog(q):
    print(h)
```

### get\_discussions\_by\_cashout

```
from beem.discussions import Query, Discussions_by_cashout
q = Query(limit=10)
for h in Discussions_by_cashout(q):
    print(h)
```

### get\_discussions\_by\_children

```
from beem.discussions import Query, Discussions_by_children
q = Query(limit=10)
for h in Discussions_by_children(q):
    print(h)
```

### get\_discussions\_by\_comments

```
from beem.discussions import Query, Discussions_by_comments
q = Query(limit=10, start_author="steemit", start_permlink="firstpost")
for h in Discussions_by_comments(q):
    print(h)
```

### get\_discussions\_by\_created

```
from beem.discussions import Query, Discussions_by_created
q = Query(limit=10)
for h in Discussions_by_created(q):
    print(h)
```

### get\_discussions\_by\_feed

```
from beem.discussions import Query, Discussions_by_feed
q = Query(limit=10, tag="steem")
for h in Discussions_by_feed(q):
    print(h)
```

### get\_discussions\_by\_hot

```
from beem.discussions import Query, Discussions_by_hot
q = Query(limit=10, tag="steem")
for h in Discussions_by_hot(q):
    print(h)
```

### get\_discussions\_by\_promoted

```
from beem.discussions import Query, Discussions_by_promoted
q = Query(limit=10, tag="steem")
for h in Discussions_by_promoted(q):
    print(h)
```

### get\_discussions\_by\_trending

```
from beem.discussions import Query, Discussions_by_trending
q = Query(limit=10, tag="steem")
for h in Discussions_by_trending(q):
    print(h)
```

### get\_discussions\_by\_votes

```
from beem.discussions import Query, Discussions_by_votes
q = Query(limit=10)
for h in Discussions_by_votes(q):
    print(h)
```

### get\_dynamic\_global\_properties

```
from beem import Steem
stm = Steem()
print(stm.get_dynamic_global_properties())
```

### get\_escrow

```
from beem.account import Account
acc = Account("gtg")
print(acc.get_escrow())
```

### get\_expiring\_vesting\_delegations

```
from beem.account import Account
acc = Account("gtg")
print(acc.get_expiring_vesting_delegations())
```

### get\_feed

```
from beem.account import Account
acc = Account("gtg")
for f in acc.get_feed():
    print(f)
```

### get\_feed\_entries

```
from beem.account import Account
acc = Account("gtg")
for f in acc.get_feed_entries():
    print(f)
```

### get\_feed\_history

```
from beem import Steem
stm = Steem()
print(stm.get_feed_history())
```

### get\_follow\_count

```
from beem.account import Account
acc = Account("gtg")
print(acc.get_follow_count())
```

### get\_followers

```
from beem.account import Account
acc = Account("gtg")
for f in acc.get_followers():
    print(f)
```

### get\_following

```
from beem.account import Account
acc = Account("gtg")
for f in acc.get_following():
    print(f)
```

### get\_hardfork\_version

```
from beem import Steem
stm = Steem()
print(stm.get_hardfork_properties()["hf_version"])
```

### get\_key\_references

```
from beem.account import Account
from beem.wallet import Wallet
acc = Account("gtg")
w = Wallet()
print(w.getAccountFromPublicKey(acc["posting"]["key_auths"][0][0]))
```

### get\_market\_history

```
from beem.market import Market
m = Market()
for t in m.market_history():
    print(t)
```

### get\_market\_history\_buckets

```
from beem.market import Market
m = Market()
for t in m.market_history_buckets():
    print(t)
```

### get\_next\_scheduled\_hardfork

```
from beem import Steem
stm = Steem()
print(stm.get_hardfork_properties())
```

### get\_open\_orders

```
from beem.market import Market
m = Market()
print(m.accountopenorders(account="gtg"))
```

### get\_ops\_in\_block

```
from beem.block import Block
b = Block(2e6, only_ops=True)
print(b)
```

### get\_order\_book

```
from beem.market import Market
m = Market()
print(m.orderbook())
```

### get\_owner\_history

```
from beem.account import Account
acc = Account("gtg")
print(acc.get_owner_history())
```

### get\_post\_discussions\_by\_payout

```
from beem.discussions import Query, Post_discussions_by_payout
q = Query(limit=10)
for h in Post_discussions_by_payout(q):
    print(h)
```

### get\_potential\_signatures

```

from beem.transactionbuilder import TransactionBuilder
from beem.blockchain import Blockchain
b = Blockchain()
block = b.get_current_block()
trx = block.json()["transactions"][0]
t = TransactionBuilder(trx)
print(t.get_potential_signatures())

```

### get\_reblogged\_by

```

from beem.account import Account
from beem.comment import Comment
acc = Account("gtg")
post = acc.get_feed(0,1)[0]
c = Comment(post["authorperm"])
for h in c.get_reblogged_by():
    print(h)

```

### get\_recent\_trades

```

from beem.market import Market
m = Market()
for t in m.recent_trades():
    print(t)

```

### get\_recovery\_request

```

from beem.account import Account
acc = Account("gtg")
print(acc.get_recovery_request())

```

### get\_replies\_by\_last\_update

```

from beem.discussions import Query, Replies_by_last_update
q = Query(limit=10, start_author="steemit", start_permlink="firstpost")
for h in Replies_by_last_update(q):
    print(h)

```

### get\_required\_signatures

```

from beem.transactionbuilder import TransactionBuilder
from beem.blockchain import Blockchain
b = Blockchain()
block = b.get_current_block()
trx = block.json()["transactions"][0]
t = TransactionBuilder(trx)
print(t.get_required_signatures())

```

### get\_reward\_fund

```
from beem import Steem
stm = Steem()
print(stm.get_reward_funds())
```

### get\_savings\_withdraw\_from

```
from beem.account import Account
acc = Account("gtg")
print(acc.get_savings_withdrawals(direction="from"))
```

### get\_savings\_withdraw\_to

```
from beem.account import Account
acc = Account("gtg")
print(acc.get_savings_withdrawals(direction="to"))
```

### get\_state

```
from beem.comment import RecentByPath
for p in RecentByPath(path="promoted"):
    print(p)
```

### get\_tags\_used\_by\_author

```
from beem.account import Account
acc = Account("gtg")
print(acc.get_tags_used_by_author())
```

### get\_ticker

```
from beem.market import Market
m = Market()
print(m.ticker())
```

### get\_trade\_history

```
from beem.market import Market
m = Market()
for t in m.trade_history():
    print(t)
```



### get\_transaction

```
from beem.blockchain import Blockchain
b = Blockchain()
print(b.get_transaction("6fde0190a97835ea6d9e651293e90c89911f933c"))
```

### get\_transaction\_hex

```
from beem.blockchain import Blockchain
b = Blockchain()
block = b.get_current_block()
trx = block.json()["transactions"][0]
print(b.get_transaction_hex(trx))
```

### get\_trending\_tags

```
from beem.discussions import Query, Trending_tags
q = Query(limit=10, start_tag="steemit")
for h in Trending_tags(q):
    print(h)
```

### get\_version

not implemented

### get\_vesting\_delegations

```
from beem.account import Account
acc = Account("gtg")
for v in acc.get_vesting_delegations():
    print(v)
```

### get\_volume

```
from beem.market import Market
m = Market()
print(m.volume24h())
```

### get\_withdraw\_routes

```
from beem.account import Account
acc = Account("gtg")
print(acc.get_withdraw_routes())
```

### get\_witness\_by\_account

```
from beem.witness import Witness
w = Witness("gtg")
print(w)
```

### get\_witness\_count

```
from beem.witness import Witnesses
w = Witnesses()
print(w.witness_count)
```

### get\_witness\_schedule

```
from beem import Steem
stm = Steem()
print(stm.get_witness_schedule())
```

### get\_witnesses

not implemented

### get\_witnesses\_by\_vote

```
from beem.witness import WitnessesRankedByVote
for w in WitnessesRankedByVote():
    print(w)
```

### lookup\_account\_names

```
from beem.account import Account
acc = Account("gtg", full=False)
print(acc.json())
```

### lookup\_accounts

```
from beem.account import Account
acc = Account("gtg")
for a in acc.get_similar_account_names(limit=100):
    print(a)
```

### lookup\_witness\_accounts

```
from beem.witness import ListWitnesses
for w in ListWitnesses():
    print(w)
```

## verify\_account\_authority

disabled and not implemented

## verify\_authority

```

from beem.transactionbuilder import TransactionBuilder
from beem.blockchain import Blockchain
b = Blockchain()
block = b.get_current_block()
trx = block.json()["transactions"][0]
t = TransactionBuilder(trx)
t.verify_authority()
print("ok")

```

## 3.7 Modules

### 3.7.1 beem Modules

#### beem.account

**class** `beem.account.Account` (*account*, *full=True*, *lazy=False*, *steem\_instance=None*)

Bases: `beem.blockchainobject.BlockchainObject`

This class allows to easily access Account data

#### Parameters

- **account\_name** (*str*) – Name of the account
- **steem\_instance** (`beem.steem.Steem`) – Steem instance
- **lazy** (*bool*) – Use lazy loading
- **full** (*bool*) – Obtain all account data including orders, positions, etc.

**Returns** Account data

**Return type** dictionary

**Raises** `beem.exceptions.AccountDoesNotExistException` – if account does not exist

Instances of this class are dictionaries that come with additional methods (see below) that allow dealing with an account and its corresponding functions.

```

>>> from beem.account import Account
>>> account = Account("gtg")
>>> print(account)
<Account gtg>
>>> print(account.balances)

```

**Note:** This class comes with its own caching function to reduce the load on the API server. Instances of this class can be refreshed with `Account.refresh()`. The cache can be cleared with `Account.clear_cache()`

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

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*) – (optional) The threshold that needs to be reached by signatures to be able to interact

**approvewitness** (*witness*, *account=None*, *approve=True*, *\*\*kwargs*)

Approve a witness

#### Parameters

- **witness** (*list*) – list of Witness name or id
- **account** (*str*) – (optional) the account to allow access to (defaults to `default_account`)

**available\_balances**

List balances of an account. This call returns instances of `beem.amount.Amount`.

**balances**

Returns all account balances as dictionary

**blog\_history** (*limit=None*, *start=-1*, *reblogs=True*, *account=None*)

**stream the blog entries done by an account in reverse time order.** Note that RPC nodes keep a limited history of entries for the user blog. Older blog posts of an account may not be available via this call due to these node limitations.

#### Parameters

- **limit** (*int*) – (optional) stream the latest *limit* blog entries. If unset (default), all available blog entries are streamed.
- **start** (*int*) – (optional) start streaming the blog entries from this index. *start=-1* (default) starts with the latest available entry.
- **reblogs** (*bool*) – (optional) if set *True* (default) reblogs / resteems are included. If set *False*, reblogs/resteems are omitted.
- **account** (*str*) – (optional) the account to stream blog entries for (defaults to `default_account`)

blog\_history\_reverse example:

```
from beem.account import Account
acc = Account("steemitblog")
for post in acc.blog_history(limit=10):
    print(post)
```

**cancel\_transfer\_from\_savings** (*request\_id*, *account=None*, *\*\*kwargs*)

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`

**claim\_reward\_balance** (*reward\_steem=0*, *reward\_sbd=0*, *reward\_vests=0*, *account=None*, *\*\*kwargs*)

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** (*str*) – Amount of STEEM you would like to claim.
- **reward\_sbd** (*str*) – Amount of SBD you would like to claim.
- **reward\_vests** (*str*) – Amount of VESTS you would like to claim.
- **account** (*str*) – The source account for the claim if not `default_account` is used.

**comment\_history** (*limit=None*, *start\_permlink=None*, *account=None*)

**stream the comments done by an account in reverse time order.** Note that RPC nodes keep a limited history of entries for the user comments. Older comments by an account may not be available via this call due to these node limitations.

#### Parameters

- **limit** (*int*) – (optional) stream the latest *limit* comments. If unset (default), all available comments are streamed.
- **start\_permlink** (*str*) – (optional) start streaming the comments from this permlink. *start\_permlink=None* (default) starts with the latest available entry.
- **account** (*str*) – (optional) the account to stream comments for (defaults to `default_account`)

`comment_history_reverse` example:

```
from beem.account import Account
acc = Account("ned")
for comment in acc.comment_history(limit=10):
    print(comment)
```

**convert** (*amount*, *account=None*, *request\_id=None*)

Convert SteemDollars to Steem (takes 3.5 days to settle)

#### Parameters

- **amount** (*float*) – amount of SBD to convert
- **account** (*str*) – (optional) the source account for the transfer if not `default_account`
- **request\_id** (*str*) – (optional) identifier for tracking the conversion

**curation\_stats** ()

Returns the curation reward of the last 24h and 7d and the average of the last 7 days

**Returns** Account curation

**Return type** dictionary

Sample output:

```
{
  '24hr': 0.0,
  '7d': 0.0,
  'avg': 0.0
}
```

**delegate\_vesting\_shares** (*to\_account*, *vesting\_shares*, *account=None*, *\*\*kwargs*)

Delegate SP to another account.

#### Parameters

- **to\_account** (*str*) – Account we are delegating shares to (delegatee).
- **vesting\_shares** (*str*) – Amount of VESTS to delegate eg. *10000 VESTS*.
- **account** (*str*) – The source account (delegator). If not specified, *default\_account* is used.

**disallow** (*foreign*, *permission='posting'*, *account=None*, *threshold=None*, *\*\*kwargs*)

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

**disapprovewitness** (*witness*, *account=None*, *\*\*kwargs*)

Disapprove a witness

#### Parameters

- **witness** (*list*) – list of Witness name or id
- **account** (*str*) – (optional) the account to allow access to (defaults to *default\_account*)

**ensure\_full** ()

Ensure that all data are loaded

**estimate\_virtual\_op\_num** (*blocktime*, *stop\_diff=0*, *max\_count=100*)

Returns an estimation of an virtual operation index for a given time or blockindex

#### Parameters

- **blocktime** (*int/datetime*) – start time or start block index from which account operation should be fetched
- **stop\_diff** (*int*) – Sets the difference between last estimation and new estimation at which the estimation stops. Must not be zero. (default is 1)
- **max\_count** (*int*) – sets the maximum number of iterations. -1 disables this (default 100)

```

utc = pytz.timezone('UTC')
start_time = utc.localize(datetime.utcnow()) - timedelta(days=7)
acc = Account("gtg")
start_op = acc.estimate_virtual_op_num(start_time)

b = Blockchain()
start_block_num = b.get_estimated_block_num(start_time)
start_op2 = acc.estimate_virtual_op_num(start_block_num)

```

```

acc = Account("gtg")
block_num = 21248120
start = t.time()
op_num = acc.estimate_virtual_op_num(block_num, stop_diff=1, max_count=10)
stop = t.time()
print(stop - start)
for h in acc.get_account_history(op_num, 0):
    block_est = h["block"]
print(block_est - block_num)

```

**feed\_history** (*limit=None, start\_author=None, start\_permlink=None, account=None*)

**stream the feed entries of an account in reverse time order.** Note that RPC nodes keep a limited history of entries for the user feed. Older entries may not be available via this call due to these node limitations.

#### Parameters

- **limit** (*int*) – (optional) stream the latest *limit* feed entries. If unset (default), all available entries are streamed.
- **start\_author** (*str*) – (optional) start streaming the replies from this author. *start\_permlink=None* (default) starts with the latest available entry. If set, *start\_permlink* has to be set as well.
- **start\_permlink** (*str*) – (optional) start streaming the replies from this permlink. *start\_permlink=None* (default) starts with the latest available entry. If set, *start\_author* has to be set as well.
- **account** (*str*) – (optional) the account to get replies to (defaults to *default\_account*)

comment\_history\_reverse example:

```

from beem.account import Account
acc = Account("ned")
for reply in acc.feed_history(limit=10):
    print(reply)

```

**follow** (*other, what=['blog'], account=None*)

Follow/Unfollow/Mute/Unmute another account's blog

#### Parameters

- **other** (*str*) – Follow this account
- **what** (*list*) – List of states to follow. ['blog'] means to follow other, [] means to unfollow/unmute other, ['ignore'] means to ignore other, (defaults to ['blog'])

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

**getSimilarAccountNames** (*limit=5*)

Deprecated, please use `get_similar_account_names`

**get\_account\_bandwidth** (*bandwidth\_type=1, account=None*)

**get\_account\_history** (*index, limit, order=-1, start=None, stop=None, use\_block\_num=True, only\_ops=[], exclude\_ops=[], raw\_output=False*)

Returns a generator for individual account transactions. This call can be used in a `for` loop.

#### Parameters

- **index** (*int*) – first number of transactions to return
- **limit** (*int*) – limit number of transactions to return
- **start** (*int/datetime*) – start number/date of transactions to return (*optional*)
- **stop** (*int/datetime*) – stop number/date of transactions to return (*optional*)
- **use\_block\_num** (*bool*) – if true, start and stop are block numbers, otherwise virtual OP count numbers.
- **only\_ops** (*array*) – Limit generator by these operations (*optional*)
- **exclude\_ops** (*array*) – Exclude these operations from generator (*optional*)
- **batch\_size** (*int*) – internal api call batch size (*optional*)
- **order** (*int*) – 1 for chronological, -1 for reverse order
- **raw\_output** (*bool*) – if False, the output is a dict, which includes all values. Otherwise, the output is list.

... **note::** `only_ops` and `exclude_ops` takes an array of strings: The full list of operation ID's can be found in `beembase.operationids.ops`. Example: `['transfer', 'vote']`

**get\_account\_votes** (*account=None*)

Returns all votes that the account has done

#### Return type list

```
>>> from beem.account import Account
>>> account = Account("beem.app")
>>> account.get_account_votes()
[]
```

**get\_balance** (*balances, symbol*)

Obtain the balance of a specific Asset. This call returns instances of `beem.amount.Amount`. Available balance types:

- “available”
- “saving”
- “reward”
- “total”

#### Parameters

- **balances** (*str*) – Defines the balance type



- **dict) symbol ( (str,) )** – Can be “SBD”, “STEEM” or “VESTS”

```
>>> from beem.account import Account
>>> account = Account("beem.app")
>>> account.get_balance("rewards", "SBD")
0.000 SBD
```

### **get\_balances ()**

Returns all account balances as dictionary

**Returns** Account balances

**Return type** dictionary

Sample output:

```
{
  'available': [102.985 STEEM, 0.008 SBD, 146273.695970 VESTS],
  'savings': [0.000 STEEM, 0.000 SBD],
  'rewards': [0.000 STEEM, 0.000 SBD, 0.000000 VESTS],
  'total': [102.985 STEEM, 0.008 SBD, 146273.695970 VESTS]
}
```

### **get\_bandwidth ()**

Returns used and allocated bandwidth

**Return type** dict

Sample output:

```
{
  'used': 0,
  'allocated': 2211037
}
```

### **get\_blog (start\_entry\_id=0, limit=100, raw\_data=False, short\_entries=False, account=None)**

Returns the list of blog entries for an account

#### **Parameters**

- **start\_entry\_id (int)** – default is 0
- **limit (int)** – default is 100
- **raw\_data (bool)** – default is False
- **short\_entries (bool)** – when set to True and raw\_data is True, get\_blog\_entries is used instead of get\_blog
- **account (str)** – When set, a different account name is used (Default is object account name)

**Return type** list

```
>>> from beem.account import Account
>>> account = Account("steemit")
>>> account.get_blog(0, 1)
[<Comment @steemit/firstpost>]
```

### **get\_blog\_authors (account=None)**

Returns a list of authors that have had their content reblogged on a given blog account

**Parameters** **account** (*str*) – When set, a different account name is used (Default is object account name)

**Return type** list

```
>>> from beem.account import Account
>>> account = Account("steemit")
>>> account.get_blog_authors()
[]
```

**get\_blog\_entries** (*start\_entry\_id=0, limit=100, raw\_data=True, account=None*)

Returns the list of blog entries for an account

**Parameters**

- **start\_entry\_id** (*int*) – default is 0
- **limit** (*int*) – default is 100
- **raw\_data** (*bool*) – default is False
- **account** (*str*) – When set, a different account name is used (Default is object account name)

**Return type** list

```
>>> from beem.account import Account
>>> account = Account("steemit")
>>> entry = account.get_blog_entries(0, 1, raw_data=True)[0]
>>> print("%s - %s - %s - %s" % (entry["author"], entry["permalink"], entry[
↳ "blog"], entry["reblog_on"]))
steemit - firstpost - steemit - 1970-01-01T00:00:00
```

**get\_conversion\_requests** (*account=None*)

Returns a list of SBD conversion request

**Parameters** **account** (*str*) – When set, a different account is used for the request (Default is object account name)

**Return type** list

```
>>> from beem.account import Account
>>> account = Account("beem.app")
>>> account.get_conversion_requests()
[]
```

**get\_creator** ()

Returns the account creator or *None* if the account was mined

**get\_curation\_reward** (*days=7*)

Returns the curation reward of the last *days* days

**Parameters** **days** (*int*) – limit number of days to be included int the return value

**get\_effective\_vesting\_shares** ()

Returns the effective vesting shares

**get\_escrow** (*escrow\_id=0, account=None*)

Returns the escrow for a certain account by id

**Parameters**

- **escrow\_id** (*int*) – Id (only pre appbase)

- **account** (*str*) – When set, a different account is used for the request (Default is object account name)

**Return type** list

```
>>> from beem.account import Account
>>> account = Account("beem.app")
>>> account.get_escrow(1234)
[]
```

**get\_expiring\_vesting\_delegations** (*after=None, limit=1000, account=None*)

Returns the expirations for vesting delegations.

:param datetime after : expiration after (only for pre appbase nodes) :param int limit: limits number of shown entries (only for pre appbase nodes) :param str account: When set, a different account is used for the request (Default is object account name)

**Return type** list

```
>>> from beem.account import Account
>>> account = Account("beem.app")
>>> account.get_expiring_vesting_delegations()
[]
```

**get\_feed** (*start\_entry\_id=0, limit=100, raw\_data=False, short\_entries=False, account=None*)

Returns a list of items in an account's feed

**Parameters**

- **start\_entry\_id** (*int*) – default is 0
- **limit** (*int*) – default is 100
- **raw\_data** (*bool*) – default is False
- **short\_entries** (*bool*) – when set to True and raw\_data is True, get\_feed\_entries is used instead of get\_feed
- **account** (*str*) – When set, a different account name is used (Default is object account name)

**Return type** list

```
>>> from beem.account import Account
>>> account = Account("steemit")
>>> account.get_feed(0, 1, raw_data=True)
[]
```

**get\_feed\_entries** (*start\_entry\_id=0, limit=100, raw\_data=True, account=None*)

Returns a list of entries in an account's feed

**Parameters**

- **start\_entry\_id** (*int*) – default is 0
- **limit** (*int*) – default is 100
- **raw\_data** (*bool*) – default is False
- **short\_entries** (*bool*) – when set to True and raw\_data is True, get\_feed\_entries is used instead of get\_feed
- **account** (*str*) – When set, a different account name is used (Default is object account name)

**Return type** list

```
>>> from beem.account import Account
>>> account = Account("steemit")
>>> account.get_feed_entries(0, 1)
[]
```

**get\_follow\_count** (*account=None*)

**get\_followers** (*raw\_name\_list=True, limit=100*)

Returns the account followers as list

**get\_following** (*raw\_name\_list=True, limit=100*)

Returns who the account is following as list

**get\_manabar** ()

“Return manabar

**get\_manabar\_recharge\_time** (*manabar, recharge\_pct\_goal=100*)

Returns the account mana recharge time in minutes

**Parameters**

- **manabar** (*dict*) – manabar dict from `get_manabar()` or `get_rc_manabar()`
- **recharge\_pct\_goal** (*float*) – mana recovery goal in percentage (default is 100)

**get\_manabar\_recharge\_time\_str** (*manabar, recharge\_pct\_goal=100*)

Returns the account manabar recharge time as string

**Parameters**

- **manabar** (*dict*) – manabar dict from `get_manabar()` or `get_rc_manabar()`
- **recharge\_pct\_goal** (*float*) – mana recovery goal in percentage (default is 100)

**get\_manabar\_recharge\_timedelta** (*manabar, recharge\_pct\_goal=100*)

Returns the account mana recharge time as timedelta object

**Parameters**

- **manabar** (*dict*) – manabar dict from `get_manabar()` or `get_rc_manabar()`
- **recharge\_pct\_goal** (*float*) – mana recovery goal in percentage (default is 100)

**get\_muters** (*raw\_name\_list=True, limit=100*)

Returns the account muters as list

**get\_muting** (*raw\_name\_list=True, limit=100*)

Returns who the account is muting as list

**get\_owner\_history** (*account=None*)

Returns the owner history of an account.

**Parameters** **account** (*str*) – When set, a different account is used for the request (Default is object account name)

**Return type** list

```
>>> from beem.account import Account
>>> account = Account("beem.app")
>>> account.get_owner_history()
[]
```

**get\_rc()**

Return RC of account

**get\_rc\_manabar()**

Returns current\_mana and max\_mana for RC

**get\_recharge\_time** (*voting\_power\_goal=100*)

Returns the account voting power recharge time in minutes

**Parameters** **voting\_power\_goal** (*float*) – voting power goal in percentage (default is 100)

**get\_recharge\_time\_str** (*voting\_power\_goal=100*)

Returns the account recharge time as string

**Parameters** **voting\_power\_goal** (*float*) – voting power goal in percentage (default is 100)

**get\_recharge\_timedelta** (*voting\_power\_goal=100*)

Returns the account voting power recharge time as timedelta object

**Parameters** **voting\_power\_goal** (*float*) – voting power goal in percentage (default is 100)

**get\_recovery\_request** (*account=None*)

Returns the recovery request for an account

**Parameters** **account** (*str*) – When set, a different account is used for the request (Default is object account name)

**Return type** list

```
>>> from beem.account import Account
>>> account = Account("beem.app")
>>> account.get_recovery_request()
[]
```

**get\_reputation()**

Returns the account reputation in the (steemit) normalized form

**get\_savings\_withdrawals** (*direction='from', account=None*)

Returns the list of savings withdrawals for an account.

**Parameters**

- **account** (*str*) – When set, a different account is used for the request (Default is object account name)
- **direction** (*str*) – Can be either from or to (only non appbase nodes)

**Return type** list

```
>>> from beem.account import Account
>>> account = Account("beem.app")
>>> account.get_savings_withdrawals()
[]
```

**get\_similar\_account\_names** (*limit=5*)

Returns limit account names similar to the current account name as a list

**Parameters** **limit** (*int*) – limits the number of accounts, which will be returned

**Returns** Similar account names as list

**Return type** list

This is a wrapper around `Blockchain.get_similar_account_names()` using the current account name as reference.

**get\_steem\_power** (*onlyOwnSP=False*)

Returns the account steem power

**get\_tags\_used\_by\_author** (*account=None*)

Returns a list of tags used by an author.

**Parameters** **account** (*str*) – When set, a different account is used for the request (Default is object account name)

**Return type** list

```
>>> from beem.account import Account
>>> account = Account("beem.app")
>>> account.get_tags_used_by_author()
[]
```

**get\_vesting\_delegations** (*start\_account="", limit=100, account=None*)

Returns the vesting delegations by an account.

**Parameters**

- **account** (*str*) – When set, a different account is used for the request (Default is object account name)
- **start\_account** (*str*) – Only used in pre-appbase nodes
- **limit** (*int*) – Only used in pre-appbase nodes

**Return type** list

```
>>> from beem.account import Account
>>> account = Account("beem.app")
>>> account.get_vesting_delegations()
[]
```

**get\_vests** (*only\_own\_vests=False*)

Returns the account vests

**get\_vote** (*comment*)

Returns a vote if the account has already voted for comment.

**Parameters** **comment** (*str/Comment*) – can be a Comment object or a authorpermlink

**get\_vote\_pct\_for\_SBD** (*sbd, voting\_power=None, steem\_power=None, not\_broadcasted\_vote=True*)

Returns the voting percentage needed to have a vote worth a given number of SBD.

If the returned number is bigger than 10000 or smaller than -10000, the given SBD value is too high for that account

**Parameters** **sbd** (*str/int/Amount*) – The amount of SBD in vote value

**get\_voting\_power** (*with\_regeneration=True*)

Returns the account voting power in the range of 0-100%

**get\_voting\_value\_SBD** (*voting\_weight=100, voting\_power=None, steem\_power=None, not\_broadcasted\_vote=True*)

Returns the account voting value in SBD

**get\_withdraw\_routes** (*account=None*)

Returns the withdraw routes for an account.

**Parameters** **account** (*str*) – When set, a different account is used for the request (Default is object account name)

**Return type** list

```
>>> from beem.account import Account
>>> account = Account("beem.app")
>>> account.get_withdraw_routes()
[]
```

**has\_voted** (*comment*)

Returns if the account has already voted for comment

**Parameters** **comment** (*str/Comment*) – can be a Comment object or a authorpermlink

**history** (*start=None, stop=None, use\_block\_num=True, only\_ops=[], exclude\_ops=[], batch\_size=1000, raw\_output=False*)

Returns a generator for individual account transactions. The earlist operation will be first. This call can be used in a for loop.

**Parameters**

- **start** (*int/datetime*) – start number/date of transactions to return (*optional*)
- **stop** (*int/datetime*) – stop number/date of transactions to return (*optional*)
- **use\_block\_num** (*bool*) – if true, start and stop are block numbers, otherwise virtual OP count numbers.
- **only\_ops** (*array*) – Limit generator by these operations (*optional*)
- **exclude\_ops** (*array*) – Exclude thse operations from generator (*optional*)
- **batch\_size** (*int*) – internal api call batch size (*optional*)
- **raw\_output** (*bool*) – if False, the output is a dict, which includes all values. Otherwise, the output is list.

... **note::** only\_ops and exclude\_ops takes an array of strings: The full list of operation ID's can be found in beembase.operationids.ops. Example: ['transfer', 'vote']

```
acc = Account("gtg")
max_op_count = acc.virtual_op_count()
# Returns the 100 latest operations
acc_op = []
for h in acc.history(start=max_op_count - 99, stop=max_op_count, use_block_
↳ num=False):
    acc_op.append(h)
len(acc_op)
```

```
100
```

```
acc = Account("test")
max_block = 21990141
# Returns the account operation inside the last 100 block. This can be empty.
acc_op = []
for h in acc.history(start=max_block - 99, stop=max_block, use_block_
↳ num=True):
```

(continues on next page)

(continued from previous page)

```
acc_op.append(h)
len(acc_op)
```

0

```
acc = Account("test")
start_time = datetime(2018, 3, 1, 0, 0, 0)
stop_time = datetime(2018, 3, 2, 0, 0, 0)
# Returns the account operation from 1.4.2018 back to 1.3.2018
acc_op = []
for h in acc.history(start=start_time, stop=stop_time):
    acc_op.append(h)
len(acc_op)
```

0

**history\_reverse** (*start=None, stop=None, use\_block\_num=True, only\_ops=[], exclude\_ops=[], batch\_size=1000, raw\_output=False*)

Returns a generator for individual account transactions. The latest operation will be first. This call can be used in a `for` loop.

#### Parameters

- **start** (*int/datetime*) – start number/date of transactions to return. If negative the `virtual_op_count` is added. (*optional*)
- **stop** (*int/datetime*) – stop number/date of transactions to return. If negative the `virtual_op_count` is added. (*optional*)
- **use\_block\_num** (*bool*) – if true, start and stop are block numbers, otherwise virtual OP count numbers.
- **only\_ops** (*array*) – Limit generator by these operations (*optional*)
- **exclude\_ops** (*array*) – Exclude these operations from generator (*optional*)
- **batch\_size** (*int*) – internal api call batch size (*optional*)
- **raw\_output** (*bool*) – if False, the output is a dict, which includes all values. Otherwise, the output is list.

... **note::** `only_ops` and `exclude_ops` takes an array of strings: The full list of operation ID's can be found in `beembase.operationids.ops`. Example: `['transfer', 'vote']`

```
acc = Account("gtg")
max_op_count = acc.virtual_op_count()
# Returns the 100 latest operations
acc_op = []
for h in acc.history_reverse(start=max_op_count, stop=max_op_count - 99, use_
    ↪block_num=False):
    acc_op.append(h)
len(acc_op)
```

100



```

max_block = 21990141
acc = Account("test")
# Returns the account operation inside the last 100 block. This can be empty.
acc_op = []
for h in acc.history_reverse(start=max_block, stop=max_block-100, use_block_
↪ num=True):
    acc_op.append(h)
len(acc_op)

```

0

```

acc = Account("test")
start_time = datetime(2018, 4, 1, 0, 0, 0)
stop_time = datetime(2018, 3, 1, 0, 0, 0)
# Returns the account operation from 1.4.2018 back to 1.3.2018
acc_op = []
for h in acc.history_reverse(start=start_time, stop=stop_time):
    acc_op.append(h)
len(acc_op)

```

0

**interest()**

Calculate interest for an account

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

Sample output:

```

{
    'interest': 0.0,
    'last_payment': datetime.datetime(2018, 1, 26, 5, 50, 27, tzinfo=<UTC>),
    'next_payment': datetime.datetime(2018, 2, 25, 5, 50, 27, tzinfo=<UTC>),
    'next_payment_duration': datetime.timedelta(-65, 52132, 684026),
    'interest_rate': 0.0
}

```

**is\_fully\_loaded**

Is this instance fully loaded / e.g. all data available?

**Return type** bool**json()****json\_metadata****mute** (*mute*, *account=None*)

Mute another account

**Parameters**

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

**name**

Returns the account name

**print\_info** (*force\_refresh=False, return\_str=False, use\_table=False, \*\*kwargs*)

Prints import information about the account

**profile**

Returns the account profile

**refresh** ()

Refresh/Obtain an account's data from the API server

**rep**

Returns the account reputation

**reply\_history** (*limit=None, start\_author=None, start\_permlink=None, account=None*)

**stream the replies to an account in reverse time order.** Note that RPC nodes keep a limited history of entries for the replies to an author. Older replies to an account may not be available via this call due to these node limitations.

#### Parameters

- **limit** (*int*) – (optional) stream the latest *limit* replies. If unset (default), all available replies are streamed.
- **start\_author** (*str*) – (optional) start streaming the replies from this author. *start\_permlink=None* (default) starts with the latest available entry. If set, *start\_permlink* has to be set as well.
- **start\_permlink** (*str*) – (optional) start streaming the replies from this permlink. *start\_permlink=None* (default) starts with the latest available entry. If set, *start\_author* has to be set as well.
- **account** (*str*) – (optional) the account to get replies to (defaults to *default\_account*)

comment\_history\_reverse example:

```
from beem.account import Account
acc = Account("ned")
for reply in acc.reply_history(limit=10):
    print(reply)
```

**reward\_balances**

**saving\_balances**

**set\_withdraw\_vesting\_route** (*to, percentage=100, account=None, auto\_vest=False, \*\*kwargs*)

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 'to' account should receive the VESTS as VESTS, or false if it should receive them as STEEM. (defaults to *False*)

**sp**

Returns the accounts Steem Power

**total\_balances****transfer** (*to*, *amount*, *asset*, *memo*=", *account*=None, *\*\*kwargs*)

Transfer an asset to another account.

**Parameters**

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

transfer example: .. code-block:: python

```
from beem.account import Account from beem import Steem active_wif = "5xxxx" stm =
Steem(keys=[active_wif]) acc = Account("test", steem_instance=stm) acc.transfer("test1", 1,
"STEEM", "test")
```

**transfer\_from\_savings** (*amount*, *asset*, *memo*, *request\_id*=None, *to*=None, *account*=None, *\*\*kwargs*)

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\_to\_savings** (*amount*, *asset*, *memo*, *to*=None, *account*=None, *\*\*kwargs*)

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, *\*\*kwargs*)

Vest STEEM

**Parameters**

- **amount** (*float*) – Amount to transfer
- **to** (*str*) – Recipient (optional) if not set equal to `account`

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

**type\_id** = 2

**unfollow** (*unfollow*, *account=None*)

Unfollow/Unmute another account's blog

#### Parameters

- **unfollow** (*str*) – Unfollow/Unmute this account
- **account** (*str*) – (optional) the account to allow access to (defaults to `default_account`)

**update\_account\_metadata** (*metadata*, *account=None*, *\*\*kwargs*)

Update an account's profile in `json_metadata`

#### Parameters

- **metadata** (*dict*) – The new metadata to use
- **account** (*str*) – (optional) the account to allow access to (defaults to `default_account`)

**update\_account\_profile** (*profile*, *account=None*, *\*\*kwargs*)

Update an account's profile in `json_metadata`

#### Parameters

- **profile** (*dict*) – The new profile to use
- **account** (*str*) – (optional) the account to allow access to (defaults to `default_account`)

Sample profile structure:

```
{
  'name': 'Holger',
  'about': 'beem Developer',
  'location': 'Germany',
  'profile_image': 'https://cl.staticflickr.com/5/4715/38733717165_
↪7070227c89_n.jpg',
  'cover_image': 'https://farm1.staticflickr.com/894/26382750057_69f5c8e568.
↪jpg',
  'website': 'https://github.com/holgern/beem'
}
```

```
from beem.account import Account
account = Account("test")
profile = account.profile
profile["about"] = "test account"
account.update_account_profile(profile)
```

**update\_memo\_key** (*key*, *account=None*, *\*\*kwargs*)

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`)

**verify\_account\_authority** (*keys*, *account=None*)

Returns true if the signers have enough authority to authorize an account.

#### Parameters

- **keys** (*list*) – public key
- **account** (*str*) – When set, a different account is used for the request (Default is object account name)

**Return type** dict

```
>>> from beem.account import Account
>>> account = Account("steemit")
>>> print(account.verify_account_authority([
→ "STM7Q2rLBqzPzFeteQZewv9Lu3NLE69fZoLeL6YK59t7UmssCBNTU"]) ["valid"])
False
```

**virtual\_op\_count** (*until=None*)

Returns the number of individual account transactions

**Return type** list

**vp**

Returns the account voting power in the range of 0-100%

**withdraw\_vesting** (*amount*, *account=None*, *\*\*kwargs*)

Withdraw VESTS from the vesting account.

#### Parameters

- **amount** (*float*) – number of VESTS to withdraw over a period of 13 weeks
- **account** (*str*) – (optional) the source account for the transfer if not `default_account`

**class** beem.account.Accounts (*name\_list*, *batch\_limit=100*, *lazy=False*, *full=True*,  
*steem\_instance=None*)

Bases: `beem.account.AccountsObject`

Obtain a list of accounts

#### Parameters

- **name\_list** (*list*) – list of accounts to fetch
- **batch\_limit** (*int*) – (optional) maximum number of accounts to fetch per call, defaults to 100
- **steem\_instance** (*steem*) – Steem() instance to use when accessing a RPCcreator = `Account(creator, steem_instance=self)`

**class** beem.account.AccountsObject

Bases: list

**printAsTable** ()

**print\_summarize\_table** (*tag\_type='Follower'*, *return\_str=False*, *\*\*kwargs*)

## beem.aes

**class** beem.aes.**AESCipher** (*key*)

Bases: object

A classical AES Cipher. Can use any size of data and any size of password thanks to padding. Also ensure the coherence and the type of the data with a unicode to byte converter.

**decrypt** (*enc*)

**encrypt** (*raw*)

**static str\_to\_bytes** (*data*)

## beem.asciichart

**class** beem.asciichart.**AsciiChart** (*height=None, width=None, offset=3, placeholder='{:.2f} ', charset='utf8'*)

Bases: object

Can be used to plot price and trade history

### Parameters

- **height** (*int*) – Height of the plot
- **width** (*int*) – Width of the plot
- **offset** (*int*) – Offset between tick strings and y-axis (default is 3)
- **placeholder** (*str*) – Defines how the numbers on the y-axes are formatted (default is '{:.2f} ')
- **charset** (*str*) – sets the charset for plotting, utf8 or ascii (default: utf8)

**adapt\_on\_series** (*series*)

Calculates the minimum, maximum and length from the given list

**Parameters series** (*list*) – time series to plot

```
from beem.asciichart import AsciiChart
chart = AsciiChart()
series = [1, 2, 3, 7, 2, -4, -2]
chart.adapt_on_series(series)
chart.new_chart()
chart.add_axis()
chart.add_curve(series)
print(str(chart))
```

**add\_axis** ()

Adds a y-axis to the canvas

```
from beem.asciichart import AsciiChart
chart = AsciiChart()
series = [1, 2, 3, 7, 2, -4, -2]
chart.adapt_on_series(series)
chart.new_chart()
chart.add_axis()
chart.add_curve(series)
print(str(chart))
```

**add\_curve** (*series*)

Add a curve to the canvas

**Parameters** **series** (*list*) – List with float data points

```
from beem.asciichart import AsciiChart
chart = AsciiChart()
series = [1, 2, 3, 7, 2, -4, -2]
chart.adapt_on_series(series)
chart.new_chart()
chart.add_axis()
chart.add_curve(series)
print(str(chart))
```

**clear\_data** ()

Clears all data

**new\_chart** (*minimum=None, maximum=None, n=None*)

Clears the canvas

```
from beem.asciichart import AsciiChart
chart = AsciiChart()
series = [1, 2, 3, 7, 2, -4, -2]
chart.adapt_on_series(series)
chart.new_chart()
chart.add_axis()
chart.add_curve(series)
print(str(chart))
```

**plot** (*series, return\_str=False*)

All in one function for plotting

```
from beem.asciichart import AsciiChart
chart = AsciiChart()
series = [1, 2, 3, 7, 2, -4, -2]
chart.plot(series)
```

**set\_parameter** (*height=None, offset=None, placeholder=None*)

Can be used to change parameter

**beem.amount**

**class** beem.amount.**Amount** (*amount, asset=None, new\_appbase\_format=True, steem\_instance=None*)

Bases: dict

This class deals with Amounts of any asset to simplify dealing with the tuple:

```
(amount, asset)
```

**Parameters**

- **args** (*list*) – Allows to deal with different representations of an amount
- **amount** (*float*) – Let's create an instance with a specific amount
- **asset** (*str*) – Let's you create an instance with a specific asset (symbol)
- **steem\_instance** (*steem.steem.Steem*) – Steem instance

**Returns** All data required to represent an Amount/Asset

**Return type** dict

**Raises** **ValueError** – if the data provided is not recognized

Way to obtain a proper instance:

- args can be a string, e.g.: “1 SBD”
- args can be a dictionary containing amount and asset\_id
- args can be a dictionary containing amount and asset
- args can be a list of a float and str (symbol)
- args can be a list of a float and a `beem.asset.Asset`
- amount and asset are defined manually

An instance is a dictionary and comes with the following keys:

- amount (float)
- symbol (str)
- asset (instance of `beem.asset.Asset`)

Instances of this class can be used in regular mathematical expressions (+-\*/%) such as:

```
from beem.amount import Amount
from beem.asset import Asset
a = Amount("1 STEEM")
b = Amount(1, "STEEM")
c = Amount("20", Asset("STEEM"))
a + b
a * 2
a += b
a /= 2.0
```

```
2.000 STEEM
2.000 STEEM
```

**amount**

Returns the amount as float

**asset**

Returns the asset as instance of `steem.asset.Asset`

**copy()**

Copy the instance and make sure not to use a reference

**json()**

**symbol**

Returns the symbol of the asset

**tuple()**

`beem.amount.check_asset(other, self)`



**beem.asset**

**class** beem.asset.**Asset** (*asset, lazy=False, full=False, steem\_instance=None*)

Bases: *beem.blockchainobject.BlockchainObject*

Deals with Assets of the network.

**Parameters**

- **Asset** (*str*) – Symbol name or object id of an asset
- **lazy** (*bool*) – Lazy loading
- **full** (*bool*) – Also obtain bitasset-data and dynamic asset dat
- **steem\_instance** (*beem.steem.Steem*) – Steem instance

**Returns** All data of an asset

---

**Note:** This class comes with its own caching function to reduce the load on the API server. Instances of this class can be refreshed with `Asset.refresh()`.

---

**asset**

**precision**

**refresh()**

Refresh the data from the API server

**symbol**

**type\_id** = 3

**beem.steem**

**class** beem.steem.**Steem** (*node="", rpcuser=None, rpcpassword=None, debug=False, data\_refresh\_time\_seconds=900, \*\*kwargs*)

Bases: *object*

Connect to the Steem network.

**Parameters**

- **node** (*str*) – Node to connect to (*optional*)
- **rpcuser** (*str*) – RPC user (*optional*)
- **rpcpassword** (*str*) – RPC password (*optional*)
- **nobroadcast** (*bool*) – Do **not** broadcast a transaction! (*optional*)
- **unsigned** (*bool*) – Do **not** sign a transaction! (*optional*)
- **debug** (*bool*) – Enable Debugging (*optional*)
- **keys** (*array, dict, string*) – Predefine the wif keys to shortcut the wallet database (*optional*)
- **wif** (*array, dict, string*) – Predefine the wif keys to shortcut the wallet database (*optional*)
- **offline** (*bool*) – Boolean to prevent connecting to network (defaults to `False`) (*optional*)

- **expiration** (*int*) – Delay in seconds until transactions are supposed to expire (*optional*) (default is 30)
- **blocking** (*str*) – Wait for broadcasted transactions to be included in a block and return full transaction (can be “head” or “irreversible”)
- **bundle** (*bool*) – Do not broadcast transactions right away, but allow to bundle operations. It is not possible to send out more than one vote operation and more than one comment operation in a single broadcast (*optional*)
- **appbase** (*bool*) – Use the new appbase rpc protocol on nodes with version 0.19.4 or higher. The settings has no effect on nodes with version of 0.19.3 or lower.
- **num\_retries** (*int*) – Set the maximum number of reconnects to the nodes before Num-RetriesReached is raised. Disabled for -1. (default is -1)
- **num\_retries\_call** (*int*) – Repeat num\_retries\_call times a rpc call on node error (default is 5)
- **timeout** (*int*) – Timeout setting for https nodes (default is 60)
- **use\_sc2** (*bool*) – When True, a steemconnect object is created. Can be used for broadcast posting op or creating hot\_links (default is False)
- **steemconnect** (*SteemConnect*) – A SteemConnect object can be set manually, set use\_sc2 to True
- **custom\_chains** (*dict*) – custom chain which should be added to the known chains

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!

If no node is provided, it will connect to default nodes of <http://geo.steem.pl>. Default settings can be changed with:

```
steem = Steem(<host>)
```

where `<host>` starts with `https://`, `ws://` or `wss://`.

The purpose of this class it to simplify interaction with Steem.

The idea is to have a class that allows to do this:

```
>>> from beem import Steem
>>> steem = Steem()
>>> print(steem.get_blockchain_version())
```

This class also deals with edits, votes and reading content.

Example for adding a custom chain:

```
from beem import Steem
stm = Steem(node=["https://mytstnet.com"], custom_chains={"MYTESTNET":
    {'chain_assets': [{'asset': 'SBD', 'id': 0, 'precision': 3, 'symbol': 'SBD'},
```

(continues on next page)

(continued from previous page)

```

        {'asset': 'STEEM', 'id': 1, 'precision': 3, 'symbol': 'STEEM
→'},
        {'asset': 'VESTS', 'id': 2, 'precision': 6, 'symbol': 'VESTS
→'}]],
    'chain_id': '79276aea5d4877d9a25892eaa01b0adf019d3e5cb12a97478df3298ccdd01674
→',
    'min_version': '0.0.0',
    'prefix': 'MTN'}
    }
)

```

**broadcast** (*tx=None*)

Broadcast a transaction to the Steem network

**Parameters** *tx* (*tx*) – Signed transaction to broadcast**chain\_params****claim\_account** (*creator, fee=None, \*\*kwargs*)

“Claim account for claimed account creation.

When fee is 0 STEEM a subsidized account is claimed and can be created later with `create_claimed_account`. The number of subsidized account is limited.

**Parameters**

- **creator** (*str*) – which account should pay the registration fee (RC or STEEM) (defaults to `default_account`)
- **fee** (*str*) – when set to 0 STEEM (default), claim account is paid by RC

**clear** ()**comment\_options** (*options, identifier, beneficiaries=[], account=None, \*\*kwargs*)

Set the comment options

**Parameters**

- **options** (*dict*) – The options to define.
- **identifier** (*str*) – Post identifier
- **beneficiaries** (*list*) – (optional) list of beneficiaries
- **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,
}

```

**connect** (*node=", rpcuser=", rpcpassword=", \*\*kwargs*)

Connect to Steem network (internal use only)

```
create_account (account_name,      creator=None,      owner_key=None,      active_key=None,
                 memo_key=None,      posting_key=None,      password=None,      addi-
                 tional_owner_keys=[], additional_active_keys=[], additional_posting_keys=[],
                 additional_owner_accounts=[], additional_active_accounts=[], addi-
                 tional_posting_accounts=[], storekeys=True, store_owner_key=False,
                 json_meta=None, **kwargs)
```

Create new account on Steem

The brainkey/password can be used to recover all generated keys (see *beemgraphenebase.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.

**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 when `nobroadcast` is set to `False`. However, it **does not import the owner key** for security reasons by default. If you set `store_owner_key` to `True`, the owner key is stored. Do NOT expect to be able to recover it from the wallet if you lose your password!

---

---

**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!

---

#### 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** (*array*) – Additional owner public keys
- **additional\_active\_keys** (*array*) – Additional active public keys
- **additional\_posting\_keys** (*array*) – Additional posting public keys
- **additional\_owner\_accounts** (*array*) – Additional owner account names
- **additional\_active\_accounts** (*array*) – Additional active account names
- **storekeys** (*bool*) – Store new keys in the wallet (default: `True`)
- **creator** (*str*) – which account should pay the registration fee (defaults to `default_account`)

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

```
create_claimed_account (account_name, creator=None, owner_key=None, active_key=None,
                        memo_key=None, posting_key=None, password=None, ad-
                        ditional_owner_keys=[], additional_active_keys=[], addi-
                        tional_posting_keys=[], additional_owner_accounts=[], ad-
                        ditional_active_accounts=[], additional_posting_accounts=[],
                        storekeys=True, store_owner_key=False, json_meta=None, com-
                        bine_with_claim_account=False, fee=None, **kwargs)
```

Create new claimed account on Steem

The brainkey/password can be used to recover all generated keys (see `beemgraphenebase.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.

**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 when `nobroadcast` is set to `False`. However, it **does not import the owner key** for security reasons by default. If you set `store_owner_key` to `True`, the owner key is stored. Do NOT expect to be able to recover it from the wallet if you lose your password!

---

---

**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!

---

### 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** (*array*) – Additional owner public keys
- **additional\_active\_keys** (*array*) – Additional active public keys
- **additional\_posting\_keys** (*array*) – Additional posting public keys
- **additional\_owner\_accounts** (*array*) – Additional owner account names
- **additional\_active\_accounts** (*array*) – Additional active account names
- **storekeys** (*bool*) – Store new keys in the wallet (default: `True`)

- **combine\_with\_claim\_account** (*bool*) – When set to True, a claim\_account operation is additionally broadcasted
- **fee** (*str*) – When combine\_with\_claim\_account is set to True, this parameter is used for the claim\_account operation
- **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\_data*, *required\_auths*=[], *required\_posting\_auths*=[], *\*\*kwargs*)

Create a custom json operation

#### Parameters

- **id** (*str*) – identifier for the custom json (max length 32 bytes)
- **json\_data** (*json*) – the json data to put into the custom\_json operation
- **required\_auths** (*list*) – (optional) required auths
- **required\_posting\_auths** (*list*) – (optional) posting auths

Note: While required\_auths and required\_posting\_auths are both optional, one of the two are needed in order to send the custom json.

```
steem.custom_json("id", "json_data",
required_posting_auths=['account'])
```

**finalizeOp** (*ops*, *account*, *permission*, *\*\*kwargs*)

This method obtains the required private keys if present in the wallet, finalizes the transaction, signs it and broadcasts it

#### Parameters

- **ops** (*operation*) – The operation (or list of operations) to broadcast
- **account** (*operation*) – The account that authorizes the operation
- **permission** (*string*) – The required permission for signing (active, owner, posting)
- **append\_to** (*object*) – This allows to provide an instance of ProposalsBuilder (see `steem.new_proposal()`) or TransactionBuilder (see `steem.new_tx()`) to specify where to put a specific operation.

---

**Note:** append\_to is exposed to every method used in the Steem class

---

---

**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!

---

---

**Note:** This uses `beem.txbuffer` as instance of `beem.transactionbuilder.TransactionBuilder`. You may want to use your own txbuffer

---

**get\_api\_methods** ()

Returns all supported api methods

**get\_apis()**

Returns all enabled apis

**get\_block\_interval** (*use\_stored\_data=True*)

Returns the block interval in seconds

**get\_blockchain\_version** (*use\_stored\_data=True*)

Returns the blockchain version

**get\_chain\_properties** (*use\_stored\_data=True*)

Return witness elected chain properties

Properties::

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

**get\_config** (*use\_stored\_data=True*)

Returns internal chain configuration.

**Parameters** *use\_stored\_data* (*bool*) – If True, the cached value is returned

**get\_current\_median\_history** (*use\_stored\_data=True*)

Returns the current median price :param bool *use\_stored\_data*: if True, stored data will be returned. If stored data are empty or old, *refresh\_data()* is used.

**get\_default\_nodes** ()

Returns the default nodes

**get\_dust\_threshold** (*use\_stored\_data=True*)

Returns the vote dust threshold

**get\_dynamic\_global\_properties** (*use\_stored\_data=True*)

This call returns the *dynamic global properties*

**Parameters** *use\_stored\_data* (*bool*) – if True, stored data will be returned. If stored data are empty or old, *refresh\_data()* is used.

**get\_feed\_history** (*use\_stored\_data=True*)

Returns the feed\_history

**Parameters** *use\_stored\_data* (*bool*) – if True, stored data will be returned. If stored data are empty or old, *refresh\_data()* is used.

**get\_hardfork\_properties** (*use\_stored\_data=True*)

Returns Hardfork and live\_time of the hardfork :param bool *use\_stored\_data*: if True, stored data will be returned. If stored data are empty or old, *refresh\_data()* is used.

**get\_median\_price** (*use\_stored\_data=True*)

Returns the current median history price as Price

**get\_network** (*use\_stored\_data=True*)

Identify the network :param bool *use\_stored\_data*: if True, stored data will be returned. If stored data are empty or old, *refresh\_data()* is used.

**Returns** Network parameters

**Return type** dict

**get\_rc\_cost** (*resource\_count*)

Returns the RC costs based on the *resource\_count*

**get\_reserve\_ratio** (*use\_stored\_data=True*)

This call returns the *reserve ratio*

**Parameters** *use\_stored\_data* (*bool*) – if True, stored data will be returned. If stored data are empty or old, *refresh\_data()* is used.

**get\_resource\_params** ()

Returns the resource parameter

**get\_resource\_pool** ()

Returns the resource pool

**get\_reward\_funds** (*use\_stored\_data=True*)

Get details for a reward fund.

**Parameters** *use\_stored\_data* (*bool*) – if True, stored data will be returned. If stored data are empty or old, *refresh\_data()* is used.

**get\_sbd\_per\_rshares** (*not\_broadcasted\_vote\_rshares=0, use\_stored\_data=True*)

Returns the current rshares to SBD ratio

**get\_steem\_per\_mvest** (*time\_stamp=None, use\_stored\_data=True*)

Returns the MVEST to STEEM ratio

**Parameters** *time\_stamp* (*int*) – (optional) if set, return an estimated STEEM per MVEST ratio for the given time stamp. If unset the current ratio is returned (default). (can also be a datetime object)

**get\_witness\_schedule** (*use\_stored\_data=True*)

Return witness elected chain properties

**hardfork**

**info** (*use\_stored\_data=True*)

Returns the global properties

**is\_connected** ()

Returns if rpc is connected

**move\_current\_node\_to\_front** ()

Returns the default node list, until the first entry is equal to the current working node url

**newWallet** (*pwd*)

Create a new wallet. This method is basically only calls `beem.wallet.create()`.

**Parameters** *pwd* (*str*) – Password to use for the new wallet

**Raises** `beem.exceptions.WalletExists` – if there is already a wallet created

**new\_tx** (*\*args, \*\*kwargs*)

Let's obtain a new txbuffer

**Returns** `int txid` id of the new txbuffer

**post** (*title, body, author=None, permalink=None, reply\_identifier=None, json\_metadata=None, comment\_options=None, community=None, app=None, tags=None, beneficiaries=None, self\_vote=False, parse\_body=False, \*\*kwargs*)

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** (*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},
        ]
    }]]
}
```

#### Parameters

- **community** (*str*) – (Optional) Name of the community we are posting into. This will also override the community specified in *json\_metadata*.
- **app** (*str*) – (Optional) Name of the app which are used for posting when not set, beem/<version> is used
- **tags** (*str/list*) – (Optional) A list of tags 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*) – (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}
]
```

#### Parameters

- **self\_vote** (*bool*) – (Optional) Upvote the post as author, right after posting.
- **parse\_body** (*bool*) – (Optional) When set to True, all mentioned users, used links and images are put into users, links and images array inside *json\_metadata*. This will override provided links, images and users inside *json\_metadata*. Hashtags will added to tags until its length is below five entries.

**prefix**

**refresh\_data** (*force\_refresh=False, data\_refresh\_time\_seconds=None*)

Read and stores steem blockchain parameters. If the last data refresh is older than `data_refresh_time_seconds`, data will be refreshed.

#### Parameters

- **force\_refresh** (*bool*) – if True, a refresh of the data is enforced
- **data\_refresh\_time\_seconds** (*float*) – set a new minimal refresh time in seconds

**rshares\_to\_sbd** (*rshares, not\_broadcasted\_vote=False, use\_stored\_data=True*)

Calculates the current SBD value of a vote

**rshares\_to\_vote\_pct** (*rshares, steem\_power=None, vests=None, voting\_power=10000, use\_stored\_data=True*)

Obtain the voting percentage for a desired rshares value for a given Steem Power or vesting shares and voting\_power. Give either `steem_power` or `vests`, not both. When the output is greater than 10000 or less than -10000, the given absolute rshares are too high.

Returns the required voting percentage (100% = 10000)

#### Parameters

- **rshares** (*number*) – desired rshares value
- **steem\_power** (*number*) – Steem Power
- **vests** (*number*) – vesting shares
- **voting\_power** (*int*) – voting power (100% = 10000)

**sbd\_symbol**

get the current chain's symbol for SBD (e.g. "TBD" on testnet)

**sbd\_to\_rshares** (*sbd, not\_broadcasted\_vote=False, use\_stored\_data=True*)

Obtain the r-shares from SBD

#### Parameters

- **sbd** (*str/int/Amount*) – SBD
- **not\_broadcasted\_vote** (*bool*) – not\_broadcasted or already broadcasted vote (True = not\_broadcasted vote). Only impactful for very high amounts of SBD. Slight modification to the value calculation, as the not\_broadcasted vote rshares decreases the reward pool.

**sbd\_to\_vote\_pct** (*sbd, steem\_power=None, vests=None, voting\_power=10000, not\_broadcasted\_vote=True, use\_stored\_data=True*)

Obtain the voting percentage for a desired SBD value for a given Steem Power or vesting shares and voting power. Give either Steem Power or vests, not both. When the output is greater than 10000 or smaller than -10000, the SBD value is too high.

Returns the required voting percentage (100% = 10000)

#### Parameters

- **sbd** (*str/int/Amount*) – desired SBD value
- **steem\_power** (*number*) – Steem Power
- **vests** (*number*) – vesting shares
- **not\_broadcasted\_vote** (*bool*) – not\_broadcasted or already broadcasted vote (True = not\_broadcasted vote). Only impactful for very high amounts of SBD. Slight

modification to the value calculation, as the not\_broadcasted vote rshares decreases the reward pool.

**set\_default\_account** (*account*)

Set the default account to be used

**set\_default\_nodes** (*nodes*)

Set the default nodes to be used

**set\_default\_vote\_weight** (*vote\_weight*)

Set the default vote weight to be used

**set\_password\_storage** (*password\_storage*)

Set the password storage mode.

When set to “no”, the password has to be provided each time. When set to “environment” the password is taken from the UNLOCK variable

When set to “keyring” the password is taken from the python keyring module. A wallet password can be stored with `python -m keyring set beem wallet password`

**Parameters password\_storage** (*str*) – can be “no”, “keyring” or “environment”

**sign** (*tx=None, wifs=[]*)

Sign a provided transaction with the 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.

**sp\_to\_rshares** (*steem\_power, voting\_power=10000, vote\_pct=10000, use\_stored\_data=True*)

Obtain the r-shares from Steem power

#### Parameters

- **steem\_power** (*number*) – Steem Power
- **voting\_power** (*int*) – voting power (100% = 10000)
- **vote\_pct** (*int*) – voting percentage (100% = 10000)

**sp\_to\_sbd** (*sp, voting\_power=10000, vote\_pct=10000, not\_broadcasted\_vote=True, use\_stored\_data=True*)

Obtain the resulting SBD vote value from Steem power :param number steem\_power: Steem Power :param int voting\_power: voting power (100% = 10000) :param int vote\_pct: voting percentage (100% = 10000) :param bool not\_broadcasted\_vote: not\_broadcasted or already broadcasted vote (True = not\_broadcasted vote). Only impactful for very big votes. Slight modification to the value calculation, as the not\_broadcasted vote rshares decreases the reward pool.

**sp\_to\_vests** (*sp, timestamp=None, use\_stored\_data=True*)

Converts SP to vests

#### Parameters

- **sp** (*float*) – Steem power to convert
- **timestamp** (*datetime*) – (Optional) Can be used to calculate the conversion rate from the past

**steem\_symbol**

get the current chains symbol for STEEM (e.g. “TESTS” on testnet)

**tx()**

Returns the default transaction buffer

**txbuffer**

Returns the currently active tx buffer

**unlock(\*args, \*\*kwargs)**

Unlock the internal wallet

**vests\_symbol**

get the current chains symbol for VESTS

**vests\_to\_rshares** (*vests*, *voting\_power*=10000, *vote\_pct*=10000, *subtract\_dust\_threshold*=True, *use\_stored\_data*=True)

Obtain the r-shares from vests

#### Parameters

- **vests** (*number*) – vesting shares
- **voting\_power** (*int*) – voting power (100% = 10000)
- **vote\_pct** (*int*) – voting percentage (100% = 10000)

**vests\_to\_sbd** (*vests*, *voting\_power*=10000, *vote\_pct*=10000, *not\_broadcasted\_vote*=True, *use\_stored\_data*=True)

Obtain the resulting SBD vote value from vests :param number vests: vesting shares :param int voting\_power: voting power (100% = 10000) :param int vote\_pct: voting percentage (100% = 10000) :param bool not\_broadcasted\_vote: not\_broadcasted or already broadcasted vote (True = not\_broadcasted vote). Only impactful for very big votes. Slight modification to the value calculation, as the not\_broadcasted vote rshares decreases the reward pool.

**vests\_to\_sp** (*vests*, *timestamp*=None, *use\_stored\_data*=True)

Converts vests to SP

#### Parameters

- **vests/float vests** (*beem.amount.Amount*) – Vests to convert
- **timestamp** (*int*) – (Optional) Can be used to calculate the conversion rate from the past

**witness\_set\_properties** (*wif*, *owner*, *props*, *use\_condenser\_api*=True)

Set witness properties

#### Parameters

- **wif** (*privkey*) – Private signing key
- **props** (*dict*) – Properties
- **owner** (*str*) – witness account name

Properties::

```
{
    "account_creation_fee": x,
    "account_subsidy_budget": x,
    "account_subsidy_decay": x,
    "maximum_block_size": x,
    "url": x,
    "sbd_exchange_rate": x,
    "sbd_interest_rate": x,
```

(continues on next page)

(continued from previous page)

```

    "new_signing_key": x
}

```

**witness\_update** (*signing\_key*, *url*, *props*, *account=None*, *\*\*kwargs*)  
Creates/updates a witness

#### Parameters

- **signing\_key** (*pubkey*) – Public signing key
- **url** (*str*) – URL
- **props** (*dict*) – Properties
- **account** (*str*) – (optional) witness account name

Properties::

```

{
    "account_creation_fee": "3.000 STEEM",
    "maximum_block_size": 65536,
    "sbd_interest_rate": 0,
}

```

## beem.nodelist

**class** beem.nodelist.NodeList

Bases: list

Returns a node list

```

from beem.nodelist import NodeList
n = NodeList()
nodes_urls = n.get_nodes()

```

**get\_nodes** (*normal=True*, *appbase=True*, *dev=False*, *testnet=False*, *testnetdev=False*, *wss=True*,  
*https=True*, *not\_working=False*)  
Returns nodes as list

#### Parameters

- **normal** (*bool*) – when True, nodes with version 0.19.5 are included
- **appbase** (*bool*) – when True, nodes with version 0.19.11 are included
- **dev** (*bool*) – when True, dev nodes with version 0.19.11 are included
- **testnet** (*bool*) – when True, testnet nodes are included
- **testnetdev** (*bool*) – When True, testnet-dev nodes are included
- **not\_working** (*bool*) – When True, all nodes including not working ones will be returned

**get\_testnet** (*testnet=True*, *testnetdev=False*)  
Returns testnet nodes

**update\_nodes** (*weights=None*, *steem\_instance=None*)  
Reads metadata from fullnodeupdate and recalculates the nodes score

**Params list/dict weight** can be used to weight the different benchmarks

**beem.steemconnect**

**class** beem.steemconnect.SteemConnect (steem\_instance=None, \*args, \*\*kwargs)

Bases: object

**Parameters** **scope** (*str*) – comma separated string with scopes login,offline,vote,comment,delete\_comment,comment\_options,custom\_json,claim\_reward\_balance

```
# Run the login_app in examples and login with a account
from beem import Steem
from beem.steemconnect import SteemConnect
from beem.comment import Comment
sc2 = SteemConnect(client_id="beem.app")
steem = Steem(steemconnect=sc2)
steem.wallet.unlock("supersecret-passphrase")
post = Comment("author/permlink", steem_instance=steem)
post.upvote(voter="test") # replace "test" with your account
```

Examples for creating steemconnect v2 urls for broadcasting in browser: .. testoutput:

```
from beem import Steem
from beem.account import Account
from beem.steemconnect import SteemConnect
from pprint import pprint
steem = Steem(nobroadcast=True, unsigned=True)
sc2 = SteemConnect(steem_instance=steem)
acc = Account("test", steem_instance=steem)
pprint(sc2.url_from_tx(acc.transfer("test1", 1, "STEEM", "test")))
```

```
'https://steemconnect.com/sign/transfer?from=test&to=test1&amount=1.000+STEEM&
↳memo=test'
```

```
from beem import Steem
from beem.transactionbuilder import TransactionBuilder
from beembase import operations
from beem.steemconnect import SteemConnect
from pprint import pprint
stm = Steem(nobroadcast=True, unsigned=True)
sc2 = SteemConnect(steem_instance=stm)
tx = TransactionBuilder(steem_instance=stm)
op = operations.Transfer(**{"from": 'test',
                           "to": 'test1',
                           "amount": '1.000 STEEM',
                           "memo": 'test'})
tx.appendOps(op)
pprint(sc2.url_from_tx(tx.json()))
```

```
'https://steemconnect.com/sign/transfer?from=test&to=test1&amount=1.000+STEEM&
↳memo=test'
```

**broadcast** (operations, username=None)

Broadcast a operations

Sample operations:

```
[
  [
```

(continues on next page)

(continued from previous page)

```

        'vote', {
            'voter': 'gandalf',
            'author': 'gtg',
            'permlink': 'steem-pressure-4-need-for-speed',
            'weight': 10000
        }
    ]
]

```

**create\_hot\_sign\_url** (*operation, params, redirect\_uri=None*)

Creates a link for broadcasting an operation

#### Parameters

- **operation** (*str*) – operation name (e.g.: vote)
- **params** (*dict*) – operation dict params
- **redirect\_uri** (*str*) – Redirects to this uri, when set

**get\_access\_token** (*code*)

**get\_login\_url** (*redirect\_uri, \*\*kwargs*)

Returns a login url for receiving token from steemconnect

#### headers

**me** (*username=None*)

Calls the me function from steemconnect

```

from beem.steemconnect import SteemConnect
sc2 = SteemConnect()
sc2.steem.wallet.unlock("supersecret-passphrase")
sc2.me(username="test")

```

**refresh\_access\_token** (*code, scope*)

**revoke\_token** (*access\_token*)

**set\_access\_token** (*access\_token*)

Is needed for broadcast() and me()

**set\_username** (*username, permission='posting'*)

Set a username for the next broadcast() or me operation() The necessary token is fetched from the wallet

**update\_user\_metadata** (*metadata*)

**url\_from\_tx** (*tx, redirect\_uri=None*)

Creates a link for broadcasting an operation

#### Parameters

- **tx** (*dict*) – includes the operation, which should be broadcast
- **redirect\_uri** (*str*) – Redirects to this uri, when set

## beem.block

**class** beem.block.**Block** (*block, only\_ops=False, only\_virtual\_ops=False, full=True, lazy=False, steem\_instance=None*)

Bases: *beem.blockchainobject.BlockchainObject*

Read a single block from the chain

#### Parameters

- **block** (*int*) – block number
- **steem\_instance** (*beem.steem.Steem*) – Steem instance
- **lazy** (*bool*) – Use lazy loading
- **only\_ops** (*bool*) – Includes only operations, when set to True (default: False)
- **only\_virtual\_ops** (*bool*) – Includes only virtual operations (default: False)

Instances of this class are dictionaries that come with additional methods (see below) that allow dealing with a block and its corresponding functions.

When `only_virtual_ops` is set to True, `only_ops` is always set to True.

In addition to the block data, the block number is stored as `self["id"]` or `self.identifier`.

```
>>> from beem.block import Block
>>> block = Block(1)
>>> print(block)
<Block 1>
```

---

**Note:** This class comes with its own caching function to reduce the load on the API server. Instances of this class can be refreshed with `Account.refresh()`.

---

#### **block\_num**

Returns the block number

#### **json()**

#### **json\_operations**

Returns all block operations as list, all dates are strings.

#### **json\_transactions**

Returns all transactions as list, all dates are strings.

#### **operations**

Returns all block operations as list

#### **ops\_statistics** (*add\_to\_ops\_stat=None*)

Returns a statistic with the occurrence of the different operation types

#### **refresh()**

Even though blocks never change, you freshly obtain its contents from an API with this method

#### **time()**

Return a datetime instance for the timestamp of this block

#### **transactions**

Returns all transactions as list

**class** `beem.block.BlockHeader` (*block, full=True, lazy=False, steem\_instance=None*)

Bases: `beem.blockchainobject.BlockchainObject`

Read a single block header from the chain

#### Parameters

- **block** (*int*) – block number



- **steem\_instance** (`beem.steem.Steem`) – Steem instance
- **lazy** (`bool`) – Use lazy loading

In addition to the block data, the block number is stored as `self["id"]` or `self.identifier`.

```
>>> from beem.block import BlockHeader
>>> block = BlockHeader(1)
>>> print(block)
<BlockHeader 1>
```

**block\_num**

Returns the block number

**json()**

**refresh()**

Even though blocks never change, you freshly obtain its contents from an API with this method

**time()**

Return a datetime instance for the timestamp of this block

## beem.blockchain

```
class beem.blockchain.Blockchain(steem_instance=None, mode='irreversible',
                                  max_block_wait_repetition=None,
                                  data_refresh_time_seconds=900)
```

Bases: `object`

This class allows to access the blockchain and read data from it

### Parameters

- **steem\_instance** (`beem.steem.Steem`) – Steem instance
- **mode** (`str`) – (default) Irreversible block (`irreversible`) or actual head block (`head`)
- **max\_block\_wait\_repetition** (`int`) – maximum wait repetition for next block where each repetition is `block_interval` long (default is 3)

This class let's you deal with blockchain related data and methods. Read blockchain related data:

Read current block and blockchain info

```
print(chain.get_current_block())
print(chain.steem.info())
```

Monitor for new blocks. When `stop` is not set, monitoring will never stop.

```
blocks = []
current_num = chain.get_current_block_num()
for block in chain.blocks(start=current_num - 99, stop=current_num):
    blocks.append(block)
len(blocks)
```

```
100
```

or each operation individually:

```
ops = []
current_num = chain.get_current_block_num()
for operation in chain.ops(start=current_num - 99, stop=current_num):
    ops.append(operation)
```

**awaitTxConfirmation** (*transaction*, *limit=10*)

Returns the transaction as seen by the blockchain after being included into a block :param dict transaction: transaction to wait for :param int limit: (optional) number of blocks to wait for the transaction (default: 10)

---

**Note:** If you want instant confirmation, you need to instantiate class:*beem.blockchain.Blockchain* with *mode="head"*, otherwise, the call will wait until confirmed in an irreversible block.

---

---

**Note:** This method returns once the blockchain has included a transaction with the **same signature**. Even though the signature is not usually used to identify a transaction, it still cannot be forfeited and is derived from the transaction contented and thus identifies a transaction uniquely.

---

**block\_time** (*block\_num*)

Returns a datetime of the block with the given block number.

**Parameters** *block\_num* (*int*) – Block number

**block\_timestamp** (*block\_num*)

Returns the timestamp of the block with the given block number as integer.

**Parameters** *block\_num* (*int*) – Block number

**blocks** (*start=None*, *stop=None*, *max\_batch\_size=None*, *threading=False*, *thread\_num=8*, *only\_ops=False*, *only\_virtual\_ops=False*)  
Yields blocks starting from *start*.

**Parameters**

- **start** (*int*) – Starting block
- **stop** (*int*) – Stop at this block
- **max\_batch\_size** (*int*) – only for appbase nodes. When not None, batch calls of are used. Cannot be combined with threading
- **threading** (*bool*) – Enables threading. Cannot be combined with batch calls
- **thread\_num** (*int*) – Defines the number of threads, when *threading* is set.
- **only\_ops** (*bool*) – Only yield operations (default: False). Cannot be combined with *only\_virtual\_ops=True*.
- **only\_virtual\_ops** (*bool*) – Only yield virtual operations (default: False)

---

**Note:** If you want instant confirmation, you need to instantiate class:*beem.blockchain.Blockchain* with *mode="head"*, otherwise, the call will wait until confirmed in an irreversible block.

---

**find\_rc\_accounts** (*name*)

Returns limit similar accounts with name as list

**Parameters** *name* (*str*) – account name to search rc params for (can also be a list of accounts)

**Returns** RC params

**Return type** list

```
>>> from beem.blockchain import Blockchain
>>> blockchain = Blockchain()
>>> ret = blockchain.find_rc_accounts(["test"])
>>> len(ret) == 1
True
```

**get\_account\_count()**

Returns the number of accounts

**get\_account\_reputations** (*start=""*, *stop=""*, *steps=1000.0*, *limit=-1*, *\*\*kwargs*)

Yields account reputation between start and stop.

**Parameters**

- **start** (*str*) – Start at this account name
- **stop** (*str*) – Stop at this account name
- **steps** (*int*) – Obtain steps ret with a single call from RPC

**get\_all\_accounts** (*start=""*, *stop=""*, *steps=1000.0*, *limit=-1*, *\*\*kwargs*)

Yields account names between start and stop.

**Parameters**

- **start** (*str*) – Start at this account name
- **stop** (*str*) – Stop at this account name
- **steps** (*int*) – Obtain steps ret with a single call from RPC

**get\_current\_block** (*only\_ops=False*, *only\_virtual\_ops=False*)

This call returns the current block

**Parameters**

- **only\_ops** (*bool*) – Returns block with operations only, when set to True (default: False)
- **only\_virtual\_ops** (*bool*) – Includes only virtual operations (default: False)

---

**Note:** The block number returned depends on the `mode` used when instantiating from this class.

---

**get\_current\_block\_num()**

This call returns the current block number

---

**Note:** The block number returned depends on the `mode` used when instantiating from this class.

---

**get\_estimated\_block\_num** (*date*, *estimateForwards=False*, *accurate=True*)

This call estimates the block number based on a given date

**Parameters** **date** (*datetime*) – block time for which a block number is estimated

---

**Note:** The block number returned depends on the `mode` used when instantiating from this class.

---

**get\_similar\_account\_names** (*name*, *limit=5*)

Returns limit similar accounts with name as list

**Parameters**

- **name** (*str*) – account name to search similars for
- **limit** (*int*) – limits the number of accounts, which will be returned

**Returns** Similar account names as list

**Return type** list

```
>>> from beem.blockchain import Blockchain
>>> blockchain = Blockchain()
>>> ret = blockchain.get_similar_account_names("test", limit=5)
>>> len(ret) == 5
True
```

**get\_transaction** (*transaction\_id*)

Returns a transaction from the blockchain

**Parameters** **transaction\_id** (*str*) – transaction\_id

**get\_transaction\_hex** (*transaction*)

Returns a hexdump of the serialized binary form of a transaction.

**Parameters** **transaction** (*dict*) – transaction

**static hash\_op** (*event*)

This method generates a hash of blockchain operation.

**is\_irreversible\_mode** ()

**ops** (*start=None*, *stop=None*, *only\_virtual\_ops=False*, *\*\*kwargs*)

Blockchain.ops() is deprecated. Please use Blockchain.stream() instead.

**ops\_statistics** (*start*, *stop=None*, *add\_to\_ops\_stat=None*, *with\_virtual\_ops=True*, *verbose=False*)

Generates statistics for all operations (including virtual operations) starting from *start*.

**Parameters**

- **start** (*int*) – Starting block
- **stop** (*int*) – Stop at this block, if set to None, the current\_block\_num is taken
- **add\_to\_ops\_stat** (*dict*) – if set, the result is added to add\_to\_ops\_stat
- **verbose** (*bool*) – if True, the current block number and timestamp is printed

This call returns a dict with all possible operations and their occurrence.

**stream** (*opNames=[]*, *raw\_ops=False*, *\*args*, *\*\*kwargs*)

Yield specific operations (e.g. comments) only

**Parameters**

- **opNames** (*array*) – List of operations to filter for
- **raw\_ops** (*bool*) – When set to True, it returns the unmodified operations (default: False)
- **start** (*int*) – Start at this block
- **stop** (*int*) – Stop at this block

- **max\_batch\_size** (*int*) – only for appbase nodes. When not None, batch calls of are used. Cannot be combined with threading
- **threading** (*bool*) – Enables threading. Cannot be combined with batch calls
- **thread\_num** (*int*) – Defines the number of threads, when *threading* is set.
- **only\_ops** (*bool*) – Only yield operations (default: False) Cannot be combined with *only\_virtual\_ops=True*
- **only\_virtual\_ops** (*bool*) – Only yield virtual operations (default: False)

The dict output is formatted such that *type* carries the operation type. Timestamp and *block\_num* are taken from the block the operation was stored in and the other keys depend on the actual operation.

---

**Note:** If you want instant confirmation, you need to instantiate class: *beem.blockchain.Blockchain* with *mode="head"*, otherwise, the call will wait until confirmed in an irreversible block.

---

output when *raw\_ops=False* is set:

```
{
  'type': 'transfer',
  'from': 'johngreenfield',
  'to': 'thundercurator',
  'amount': '0.080 SBD',
  'memo': 'https://steemit.com/lofi/@johngreenfield/lofi-joji-yeah-right',
  '_id': '6d4c5f2d4d8ef1918acae4a8dce34f9da384786',
  'timestamp': datetime.datetime(2018, 5, 9, 11, 23, 6, tzinfo=<UTC>),
  'block_num': 22277588, 'trx_num': 35, 'trx_id':
  ↳ 'cf11b2ac8493c71063ec121b2e8517ab1e0e6bea'
}
```

output when *raw\_ops=True* is set:

```
{
  'block_num': 22277588,
  'op':
    [
      'transfer',
      {
        'from': 'johngreenfield', 'to': 'thundercurator',
        'amount': '0.080 SBD',
        'memo': 'https://steemit.com/lofi/@johngreenfield/lofi-
        ↳ joji-yeah-right'
      }
    ],
  'timestamp': datetime.datetime(2018, 5, 9, 11, 23, 6, tzinfo=<UTC>)
}
```

**wait\_for\_and\_get\_block** (*block\_number*, *blocks\_waiting\_for=None*, *only\_ops=False*,  
*only\_virtual\_ops=False*, *block\_number\_check\_cnt=-1*,  
*last\_current\_block\_num=None*)

Get the desired block from the chain, if the current head block is smaller (for both head and irreversible) then we wait, but a maximum of *blocks\_waiting\_for \* max\_block\_wait\_repetition* time before failure.

#### Parameters

- **block\_number** (*int*) – desired block number

- **blocks\_waiting\_for** (*int*) – difference between block\_number and current head and defines how many blocks we are willing to wait, positive int (default: None)
- **only\_ops** (*bool*) – Returns blocks with operations only, when set to True (default: False)
- **only\_virtual\_ops** (*bool*) – Includes only virtual operations (default: False)
- **block\_number\_check\_cnt** (*int*) – limit the number of retries when greater than -1
- **last\_current\_block\_num** (*int*) – can be used to reduce the number of get\_current\_block\_num() api calls

**class** beem.blockchain.Pool (*thread\_count, batch\_mode=True, exception\_handler=<function default\_handler>*)

Bases: object

Pool of threads consuming tasks from a queue

**abort** (*block=False*)

Tell each worker that its done working

**alive** ()

Returns True if any threads are currently running

**done** ()

Returns True if not tasks are left to be completed

**enqueue** (*func, \*args, \*\*kwargs*)

Add a task to the queue

**idle** ()

Returns True if all threads are waiting for work

**join** ()

Wait for completion of all the tasks in the queue

**results** (*sleep\_time=0*)

Get the set of results that have been processed, repeatedly call until done

**run** (*block=False*)

Start the threads, or restart them if you've aborted

**class** beem.blockchain.Worker (*name, queue, results, abort, idle, exception\_handler*)

Bases: threading.Thread

Thread executing tasks from a given tasks queue

**run** ()

Thread work loop calling the function with the params

beem.blockchain.default\_handler (*name, exception, \*args, \*\*kwargs*)

## beem.blockchainobject

**class** beem.blockchainobject.BlockchainObject (*data, klass=None, space\_id=1, object\_id=None, lazy=False, use\_cache=True, id\_item=None, steem\_instance=None, \*args, \*\*kwargs*)

Bases: dict

**cache** ()

**static clear\_cache** ()

```

clear_cache_from_expired_items()
get_cache_auto_clean()
get_cache_expiration()
getcache(id)
iscached(id)
items() → a set-like object providing a view on D's items
json()
set_cache_auto_clean(auto_clean)
set_cache_expiration(expiration)
space_id = 1
test_valid_objectid(i)
testid(id)
type_id = None
type_ids = []
class beem.blockchainobject.ObjectCache(initial_data={},          default_expiration=10,
                                         auto_clean=True)
    Bases: dict
    clear_expired_items()
    get(k[, d]) → D[k] if k in D, else d. d defaults to None.

```

## beem.comment

```

class beem.comment.Comment(authorperm, full=True, lazy=False, steem_instance=None)
    Bases: beem.blockchainobject.BlockchainObject

```

Read data about a Comment/Post in the chain

### Parameters

- **authorperm** (*str*) – identifier to post/comment in the form of @author/permlink
- **steem\_instance** (*steem*) – Steem() instance to use when accessing a RPC

```

>>> from beem.comment import Comment
>>> from beem.account import Account
>>> acc = Account("gtg")
>>> authorperm = acc.get_blog(limit=1)[0]["authorperm"]
>>> c = Comment(authorperm)
>>> postdate = c["created"]
>>> postdate_str = c.json()["created"]

```

**author**

**authorperm**

**body**

**category**

**curation\_penalty\_compensation\_SBD()**

Returns The required post payout amount after 15 minutes which will compensate the curation penalty, if voting earlier than 15 minutes

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

Delete an existing post/comment

**Parameters**

- **account** (*str*) – (optional) Account to use for deletion. If *account* is not defined, the *default\_account* will be taken or a *ValueError* will be raised.
- **identifier** (*str*) – (optional) Identifier for the post to delete. Takes the form *@author/permlink*. By default the current post will be used.

**Note:** a post/comment can only be deleted as long as it has no replies and no positive rshares on it.

**depth****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*)

**estimate\_curation\_SBD** (*vote\_value\_SBD, estimated\_value\_SBD=None*)

Estimates curation reward

**Parameters**

- **vote\_value\_SBD** (*float*) – The vote value in SBD for which the curation should be calculated
- **estimated\_value\_SBD** (*float*) – When set, this value is used for calculate the curation. When not set, the current post value is used.

**get\_all\_replies** (*parent=None*)

Returns all content replies

**get\_author\_rewards** ()

Returns the author rewards.

Example::

```
{
  'pending_rewards': True,
  'payout_SP': 0.912 STEEM,
  'payout_SBD': 3.583 SBD,
```

(continues on next page)



(continued from previous page)

```

    'total_payout_SBD': 7.166 SBD
}

```

**get\_beneficiaries\_pct()**

Returns the sum of all post beneficiaries in percentage

**get\_curation\_penalty()** (*vote\_time=None*)

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

**Parameters** *vote\_time* (*datetime*) – A vote time can be given and the curation penalty is calculated regarding the given time (default is None) When set to None, the current date is used.

**Returns** Float number between 0 and 1 (0.0 -> no penalty, 1.0 -> 100 % curation penalty)

**Return type** float

**get\_curation\_rewards()** (*pending\_payout\_SBD=False, pending\_payout\_value=None*)

Returns the curation rewards.

**Parameters**

- **pending\_payout\_SBD** (*bool*) – If True, the rewards are returned in SBD and not in STEEM (default is False)
- **pending\_payout\_value** (*float/str*) – When not None this value instead of the current value is used for calculating the rewards

*pending\_rewards* is True when the post is younger than 7 days. *unclaimed\_rewards* is the amount of curation\_rewards that goes to the author (self-vote or votes within the first 30 minutes). *active\_votes* contains all voter with their curation reward.

Example::

```

{
    'pending_rewards': True, 'unclaimed_rewards': 0.245 STEEM,
    'active_votes': {
        'leprechaun': 0.006 STEEM, 'timcliff': 0.186 STEEM,
        'st3llar': 0.000 STEEM, 'crokkon': 0.015 STEEM, 'feedyourminnows': 0.
↪003 STEEM,
        'isnochys': 0.003 STEEM, 'loshcat': 0.001 STEEM, 'greenorange': 0.000_
↪STEEM,
        'qustodian': 0.123 STEEM, 'jpphotography': 0.002 STEEM, 'thinkingmind
↪': 0.001 STEEM,
        'oups': 0.006 STEEM, 'mattockfs': 0.001 STEEM, 'holger80': 0.003_
↪STEEM, 'michaelizer': 0.004 STEEM,
        'flugschwein': 0.010 STEEM, 'ulisessabeque': 0.000 STEEM, 'hakancelik
↪': 0.002 STEEM, 'sbi2': 0.008 STEEM,
        'zcool': 0.000 STEEM, 'steemhq': 0.002 STEEM, 'rowdiya': 0.000 STEEM,
↪'curator-tier-1-2': 0.012 STEEM
    }
}

```

**get\_parent()** (*children=None*)

Returns the parent post width depth == 0

**get\_reblogged\_by()** (*identifier=None*)

Shows in which blogs this post appears

**get\_replies()** (*raw\_data=False, identifier=None*)

Returns content replies

**Parameters** `raw_data` (*bool*) – When set to False, the replies will be returned as Comment class objects

**get\_rewards()**

Returns the total\_payout, author\_payout and the curator payout in SBD. When the payout is still pending, the estimated payout is given out.

Example::

```
{
  'total_payout': 9.956 SBD,
  'author_payout': 7.166 SBD,
  'curator_payout': 2.790 SBD
}
```

**get\_vote\_with\_curation** (*voter=None, raw\_data=False, pending\_payout\_value=None*)

Returns vote for voter. Returns None, if the voter cannot be found in *active\_votes*.

**Parameters**

- **voter** (*str*) – Voter for which the vote should be returned
- **raw\_data** (*bool*) – If True, the raw data are returned
- **pending\_payout\_SBD** (*float/str*) – When not None this value instead of the current value is used for calculating the rewards

**get\_votes** (*raw\_data=False*)

Returns all votes as ActiveVotes object

**id**

**is\_comment()**

Returns True if post is a comment

**is\_main\_post()**

Returns True if main post, and False if this is a comment (reply).

**is\_pending()**

Return if the payout is pending (the post/comment is younger than 7 days)

**json()**

**json\_metadata**

**parent\_author**

**parent\_permalink**

**permalink**

**refresh()**

**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)

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

Resteen a post

#### Parameters

- **identifier** (*str*) – post identifier (@<account>/<permlink>)
- **account** (*str*) – (optional) the account to allow access to (defaults to `default_account`)

**reward**

Return the estimated total SBD reward.

**time\_elapsed** ()

Return a `timedelta` on how old the post is.

**title**

**type\_id** = 8

**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, account=None, identifier=None, \*\*kwargs*)

Vote for a post

#### Parameters

- **weight** (*float*) – Voting weight. Range: -100.0 - +100.0.
- **account** (*str*) – (optional) Account to use for voting. If `account` is not defined, the `default_account` will be used or a `ValueError` will be raised
- **identifier** (*str*) – Identifier for the post to vote. Takes the form @author/permlink.

**class** beem.comment.RecentByPath (*path='promoted', category=None, lazy=False, full=True, steem\_instance=None*)

Bases: `list`

Obtain a list of votes for an account

#### Parameters

- **account** (*str*) – Account name
- **steem\_instance** (*steem*) – `Steem()` instance to use when accessing a RPC

**class** beem.comment.RecentReplies (*author, skip\_own=True, lazy=False, full=True, steem\_instance=None*)

Bases: `list`

Obtain a list of recent replies

#### Parameters

- **author** (*str*) – author
- **skip\_own** (*bool*) – (optional) Skip replies of the author to him/herself. Default: `True`
- **steem\_instance** (*steem*) – `Steem()` instance to use when accessing a RPC

## beem.conveyor

**class** `beem.conveyor.Conveyor` (*url*='https://conveyor.steemit.com', *steem\_instance*=None)

Bases: `object`

Class to access Steemit Conveyor instances: <https://github.com/steemit/conveyor>

Description from the official documentation:

- Feature flags: “Feature flags allows our apps (condenser mainly) to hide certain features behind flags.”
- User data: “Conveyor is the central point for storing sensitive user data (email, phone, etc). No other services should store this data and should instead query for it here every time.”
- User tags: “Tagging mechanism for other services, allows defining and assigning tags to accounts (or other identifiers) and querying for them.”

Not contained in the documentation, but implemented and working:

- Draft handling: saving, listing and removing post drafts consisting of a post title and a body.

The underlying RPC authentication and request signing procedure is described here: <https://github.com/steemit/rpc-auth>

**get\_feature\_flag** (*account*, *flag*, *signing\_account*=None)

Test if a specific feature flag is set for an account. The request has to be signed by the requested account or an admin account.

### Parameters

- **account** (*str*) – requested account
- **flag** (*str*) – flag to be tested
- **signing\_account** (*str*) – (optional) account to sign the request. If unset, *account* is used.

Example:

```
from beem import Steem
from beem.conveyor import Conveyor
s = Steem(keys=["5JPOSTINGKEY"])
c = Conveyor(steem_instance=s)
print(c.get_feature_flag('accountname', 'accepted_tos'))
```

**get\_feature\_flags** (*account*, *signing\_account*=None)

Get the account’s feature flags. The request has to be signed by the requested account or an admin account.

### Parameters

- **account** (*str*) – requested account
- **signing\_account** (*str*) – (optional) account to sign the request. If unset, *account* is used.

Example:

```
from beem import Steem
from beem.conveyor import Conveyor
s = Steem(keys=["5JPOSTINGKEY"])
c = Conveyor(steem_instance=s)
print(c.get_feature_flags('accountname'))
```

**get\_user\_data** (*account*, *signing\_account*=None)

Get the account's email address and phone number. The request has to be signed by the requested account or an admin account.

**Parameters**

- **account** (*str*) – requested account
- **signing\_account** (*str*) – (optional) account to sign the request. If unset, *account* is used.

Example:

```
from beem import Steem
from beem.conveyor import Conveyor
s = Steem(keys=["5JPOSTINGKEY"])
c = Conveyor(steem_instance=s)
print(c.get_user_data('accountname'))
```

**healthcheck** ()

Get the Conveyor status

Sample output:

```
{
  'ok': True, 'version': '1.1.1-4d28e36-1528725174',
  'date': '2018-07-21T12:12:25.502Z'
}
```

**list\_drafts** (*account*)

List all saved drafts from *account*

**Parameters** **account** (*str*) – requested account

Sample output:

```
{
  'jsonrpc': '2.0', 'id': 2, 'result': [
    {'title': 'draft-title', 'body': 'draft-body',
     'uuid': '06497ele-ac30-48cb-a069-27e1672924c9'}
  ]
}
```

**prehash\_message** (*timestamp*, *account*, *method*, *params*, *nonce*)

Prepare a hash for the Conveyor API request with SHA256 according to <https://github.com/steemit/rpc-auth> Hashing of *second* is then done inside *ecdsasig.sign\_message()*.

**Parameters**

- **timestamp** (*str*) – valid iso8601 datetime ending in “Z”
- **account** (*str*) – valid steem blockchain account name
- **method** (*str*) – Conveyor method name to be called
- **param** (*bytes*) – base64 encoded request parameters
- **nonce** (*bytes*) – random 8 bytes

**remove\_draft** (*account*, *uuid*)

Remove a draft from the Conveyor database

**Parameters**

- **account** (*str*) – requested account
- **uuid** (*str*) – draft identifier as returned from *list\_drafts*

**save\_draft** (*account, title, body*)

Save a draft in the Conveyor database

**Parameters**

- **account** (*str*) – requested account
- **title** (*str*) – draft post title
- **body** (*str*) – draft post body

**set\_user\_data** (*account, params, signing\_account=None*)

Set the account's email address and phone number. The request has to be signed by an admin account.

**Parameters**

- **account** (*str*) – requested account
- **param** (*dict*) – user data to be set
- **signing\_account** (*str*) – (optional) account to sign the request. If unset, *account* is used.

Example:

```
from beem import Steem
from beem.conveyor import Conveyor
s = Steem(keys=["5JADMINPOSTINGKEY"])
c = Conveyor(steem_instance=s)
userdata = {'email': 'foo@bar.com', 'phone': '+123456789'}
c.set_user_data('accountname', userdata, 'adminaccountname')
```

## beem.discussions

**class** beem.discussions.**Comment\_discussions\_by\_payout** (*discussion\_query, lazy=False, steem\_instance=None*)

Bases: list

Get comment\_discussions\_by\_payout

**Parameters**

- **discussion\_query** (beem.discussions.Query) – Defines the parameter for searching posts
- **steem\_instance** (beem.steem.Steem) – Steem instance

```
from beem.discussions import Query, Comment_discussions_by_payout
q = Query(limit=10)
for h in Comment_discussions_by_payout(q):
    print(h)
```

**class** beem.discussions.**Discussions** (*lazy=False, steem\_instance=None*)

Bases: object

Get Discussions

**Parameters** **steem\_instance** (beem.steem.Steem) – Steem instance

**get\_discussions** (*discussion\_type*, *discussion\_query*, *limit=1000*)

Get Discussions

#### Parameters

- **discussion\_type** (*str*) – Defines the used discussion query
- **discussion\_query** (*beem.discussions.Query*) –

```
from beem.discussions import Query, Discussions
query = Query(limit=51, tag="steemit")
discussions = Discussions()
count = 0
for d in discussions.get_discussions("tags", query, limit=200):
    print ("%d. " % (count + 1)) + str(d)
    count += 1
```

**class** *beem.discussions.Discussions\_by\_active* (*discussion\_query*, *lazy=False*,  
*steem\_instance=None*)

Bases: list

*get\_discussions\_by\_active*

#### Parameters

- **discussion\_query** (*beem.discussions.Query*) – Defines the parameter searching posts
- **steem\_instance** (*steem*) – Steem() instance to use when accessing a RPC

```
from beem.discussions import Query, Discussions_by_active
q = Query(limit=10)
for h in Discussions_by_active(q):
    print (h)
```

**class** *beem.discussions.Discussions\_by\_author\_before\_date* (*author="*,  
*start\_permlink="*,  
*before\_date='1970-01-01T00:00:00'*,  
*limit=100*, *lazy=False*,  
*steem\_instance=None*)

Bases: list

Get Discussions by author before date

---

**Note:** To retrieve discussions before date, the time of creation of the discussion @author/start\_permlink must be older than the specified before\_date parameter.

---

#### Parameters

- **author** (*str*) – Defines the author (*required*)
- **start\_permlink** (*str*) – Defines the permlink of a starting discussion
- **before\_date** (*str*) – Defines the before date for query
- **limit** (*int*) – Defines the limit of discussions
- **steem\_instance** (*beem.steem.Steem*) – Steem instance

```
class beem.discussions.Discussions_by_blog (discussion_query, lazy=False,
                                           steem_instance=None)
```

Bases: list

Get discussions by blog

#### Parameters

- **discussion\_query** (beem.discussions.Query) – Defines the parameter searching posts, tag must be set to a username
- **steem\_instance** (beem.steem.Steem) – Steem instance

```
from beem.discussions import Query, Discussions_by_blog
q = Query(limit=10)
for h in Discussions_by_blog(q):
    print (h)
```

```
class beem.discussions.Discussions_by_cashout (discussion_query, lazy=False,
                                              steem_instance=None)
```

Bases: list

Get discussions\_by\_cashout. This query seems to be broken at the moment. The output is always empty.

#### Parameters

- **discussion\_query** (beem.discussions.Query) – Defines the parameter searching posts
- **steem\_instance** (beem.steem.Steem) – Steem instance

```
from beem.discussions import Query, Discussions_by_cashout
q = Query(limit=10)
for h in Discussions_by_cashout(q):
    print (h)
```

```
class beem.discussions.Discussions_by_children (discussion_query, lazy=False,
                                              steem_instance=None)
```

Bases: list

Get discussions by children

#### Parameters

- **discussion\_query** (beem.discussions.Query) – Defines the parameter searching posts
- **steem\_instance** (beem.steem.Steem) – Steem instance

```
from beem.discussions import Query, Discussions_by_children
q = Query(limit=10)
for h in Discussions_by_children(q):
    print (h)
```

```
class beem.discussions.Discussions_by_comments (discussion_query, lazy=False,
                                              steem_instance=None)
```

Bases: list

Get discussions by comments

#### Parameters



- **discussion\_query** (`beem.discussions.Query`) – Defines the parameter searching posts, `start_author` and `start_permlink` must be set.
- **steem\_instance** (`beem.steem.Steem`) – Steem instance

```
from beem.discussions import Query, Discussions_by_comments
q = Query(limit=10, start_author="steemit", start_permlink="firstpost")
for h in Discussions_by_comments(q):
    print(h)
```

```
class beem.discussions.Discussions_by_created(discussion_query,          lazy=False,
                                              steem_instance=None)
```

Bases: list

Get discussions\_by\_created

#### Parameters

- **discussion\_query** (`beem.discussions.Query`) – Defines the parameter for searching posts
- **steem\_instance** (`beem.steem.Steem`) – Steem instance

```
from beem.discussions import Query, Discussions_by_created
q = Query(limit=10)
for h in Discussions_by_created(q):
    print(h)
```

```
class beem.discussions.Discussions_by_feed(discussion_query,          lazy=False,
                                           steem_instance=None)
```

Bases: list

Get discussions by feed

#### Parameters

- **discussion\_query** (`beem.discussions.Query`) – Defines the parameter searching posts, `tag` must be set to a username
- **steem\_instance** (`beem.steem.Steem`) – Steem instance

```
from beem.discussions import Query, Discussions_by_feed
q = Query(limit=10, tag="steem")
for h in Discussions_by_feed(q):
    print(h)
```

```
class beem.discussions.Discussions_by_hot(discussion_query,          lazy=False,
                                           steem_instance=None)
```

Bases: list

Get discussions by hot

#### Parameters

- **discussion\_query** (`beem.discussions.Query`) – Defines the parameter searching posts
- **steem\_instance** (`beem.steem.Steem`) – Steem instance

```
from beem.discussions import Query, Discussions_by_hot
q = Query(limit=10, tag="steem")
for h in Discussions_by_hot(q):
    print(h)
```

```
class beem.discussions.Discussions_by_promoted (discussion_query,          lazy=False,
                                              steem_instance=None)
```

Bases: list

Get discussions by promoted

#### Parameters

- **discussion\_query** (beem.discussions.Query) – Defines the parameter searching posts
- **steem\_instance** (beem.steem.Steem) – Steem instance

```
from beem.discussions import Query, Discussions_by_promoted
q = Query(limit=10, tag="steem")
for h in Discussions_by_promoted(q):
    print(h)
```

```
class beem.discussions.Discussions_by_trending (discussion_query,          lazy=False,
                                              steem_instance=None)
```

Bases: list

Get Discussions by trending

#### Parameters

- **discussion\_query** (beem.discussions.Query) – Defines the parameter for searching posts
- **steem\_instance** (beem.steem.Steem) – Steem instance

```
from beem.discussions import Query, Discussions_by_trending
q = Query(limit=10, tag="steem")
for h in Discussions_by_trending(q):
    print(h)
```

```
class beem.discussions.Discussions_by_votes (discussion_query,          lazy=False,
                                              steem_instance=None)
```

Bases: list

Get discussions\_by\_votes

#### Parameters

- **discussion\_query** (beem.discussions.Query) – Defines the parameter searching posts
- **steem\_instance** (beem.steem.Steem) – Steem instance

```
from beem.discussions import Query, Discussions_by_votes
q = Query(limit=10)
for h in Discussions_by_votes(q):
    print(h)
```

```
class beem.discussions.Post_discussions_by_payout (discussion_query,          lazy=False,
                                              steem_instance=None)
```

Bases: list

Get post\_discussions\_by\_payout

#### Parameters

- **discussion\_query** (beem.discussions.Query) – Defines the parameter for searching posts

- **steem\_instance** (`beem.steem.Steem`) – Steem instance

```
from beem.discussions import Query, Post_discussions_by_payout
q = Query(limit=10)
for h in Post_discussions_by_payout(q):
    print(h)
```

```
class beem.discussions.Query(limit=0, tag="", truncate_body=0, filter_tags=[], select_authors=[], select_tags=[], start_author=None, start_permlink=None, start_tag=None, parent_author=None, parent_permlink=None, start_parent_author=None, before_date=None, author=None)
```

Bases: dict

Query to be used for all discussion queries

#### Parameters

- **limit** (*int*) – limits the number of posts
- **tag** (*str*) – tag query
- **truncate\_body** (*int*) –
- **filter\_tags** (*array*) –
- **select\_authors** (*array*) –
- **select\_tags** (*array*) –
- **start\_author** (*str*) –
- **start\_permlink** (*str*) –
- **start\_tag** (*str*) –
- **parent\_author** (*str*) –
- **parent\_permlink** (*str*) –
- **start\_parent\_author** (*str*) –
- **before\_date** (*str*) –
- **author** (*str*) – Author (see Discussions\_by\_author\_before\_date)

```
from beem.discussions import Query
query = Query(limit=10, tag="steemit")
```

```
class beem.discussions.Replies_by_last_update(discussion_query, steem_instance=None, lazy=False)
```

Bases: list

Returns a list of replies by last update

#### Parameters

- **discussion\_query** (`beem.discussions.Query`) – Defines the parameter searching posts `start_parent_author` and `start_permlink` must be set.
- **steem\_instance** (`beem.steem.Steem`) – Steem instance

```
from beem.discussions import Query, Replies_by_last_update
q = Query(limit=10, start_parent_author="steemit", start_permlink="firstpost")
```

(continues on next page)

(continued from previous page)

```
for h in Replies_by_last_update(q):  
    print(h)
```

**class** beem.discussions.**Trending\_tags** (*discussion\_query*, *lazy=False*, *steem\_instance=None*)  
Bases: list

Returns the list of trending tags.

**Parameters**

- **discussion\_query** (beem.discussions.Query) – Defines the parameter searching posts, start\_tag can be set.
- **steem\_instance** (beem.steem.Steem) – Steem instance

```
from beem.discussions import Query, Trending_tags  
q = Query(limit=10, start_tag="")  
for h in Trending_tags(q):  
    print(h)
```

## beem.exceptions

**exception** beem.exceptions.**AccountDoesNotExistException**  
Bases: Exception

The account does not exist

**exception** beem.exceptions.**AccountExistsException**  
Bases: Exception

The requested account already exists

**exception** beem.exceptions.**AssetDoesNotExistException**  
Bases: Exception

The asset does not exist

**exception** beem.exceptions.**BatchedCallsNotSupported**  
Bases: Exception

Batch calls do not work

**exception** beem.exceptions.**BlockDoesNotExistException**  
Bases: Exception

The block does not exist

**exception** beem.exceptions.**BlockWaitTimeExceeded**  
Bases: Exception

Wait time for new block exceeded

**exception** beem.exceptions.**ContentDoesNotExistException**  
Bases: Exception

The content does not exist

**exception** beem.exceptions.**InsufficientAuthorityError**  
Bases: Exception

The transaction requires signature of a higher authority

**exception** `beem.exceptions.InvalidAssetException`

Bases: `Exception`

An invalid asset has been provided

**exception** `beem.exceptions.InvalidMemoKeyException`

Bases: `Exception`

Memo key in message is invalid

**exception** `beem.exceptions.InvalidMessageSignature`

Bases: `Exception`

The message signature does not fit the message

**exception** `beem.exceptions.InvalidWifError`

Bases: `Exception`

The provided private Key has an invalid format

**exception** `beem.exceptions.MissingKeyError`

Bases: `Exception`

A required key couldn't be found in the wallet

**exception** `beem.exceptions.NoWalletException`

Bases: `Exception`

No Wallet could be found, please use `steem.wallet.create()` to create a new wallet

**exception** `beem.exceptions.NoWriteAccess`

Bases: `Exception`

Cannot store to sqlite3 database due to missing write access

**exception** `beem.exceptions.OfflineHasNoRPCEException`

Bases: `Exception`

When in offline mode, we don't have RPC

**exception** `beem.exceptions.RPCConnectionRequired`

Bases: `Exception`

An RPC connection is required

**exception** `beem.exceptions.VestingBalanceDoesNotExistsException`

Bases: `Exception`

Vesting Balance does not exist

**exception** `beem.exceptions.VoteDoesNotExistsException`

Bases: `Exception`

The vote does not exist

**exception** `beem.exceptions.VotingInvalidOnArchivedPost`

Bases: `Exception`

The transaction requires signature of a higher authority

**exception** `beem.exceptions.WalletExists`

Bases: `Exception`

A wallet has already been created and requires a password to be unlocked by means of `steem.wallet.unlock()`.

**exception** beem.exceptions.WalletLocked

Bases: Exception

Wallet is locked

**exception** beem.exceptions.WitnessDoesNotExistsException

Bases: Exception

The witness does not exist

**exception** beem.exceptions.WrongMasterPasswordException

Bases: Exception

The password provided could not properly unlock the wallet

**exception** beem.exceptions.WrongMemoKey

Bases: Exception

The memo provided is not equal the one on the blockchain

## beem.imageuploader

**class** beem.imageuploader.ImageUploader (*base\_url='https://steemitimages.com',  
challenge='ImageSigningChallenge',  
steem\_instance=None*)

Bases: object

**upload** (*image, account, image\_name=None*)

Uploads an image

### Parameters

- **image** (*str/bytes*) – path to the image or image in bytes representation which should be uploaded
- **account** (*str*) – Account which is used to upload. A posting key must be provided.
- **image\_name** (*str*) – optional

```
from beem import Steem
from beem.imageuploader import ImageUploader
stm = Steem(keys=["5xxx"]) # private posting key
iu = ImageUploader(steem_instance=stm)
iu.upload("path/to/image.png", "account_name") # "private posting key belongs_
↳to account_name
```

## beem.instance

**class** beem.instance.SharedInstance

Bases: object

Singelton for the Steem Instance

**config** = {}

**instance** = None

beem.instance.clear\_cache()

Clear Caches

`beem.instance.set_shared_config(config)`

This allows to set a config that will be used when calling `shared_steem_instance` and allows to define the configuration without requiring to actually create an instance

`beem.instance.set_shared_steem_instance(steem_instance)`

This method allows us to override default steem instance for all users of `SharedInstance.instance`.

**Parameters** `steem_instance` (`beem.steem.Steem`) – Steem instance

`beem.instance.shared_steem_instance()`

This method will initialize `SharedInstance.instance` and return it. The purpose of this method is to have offer single default steem instance that can be reused by multiple classes.

```
from beem.account import Account
from beem.instance import shared_steem_instance

account = Account("test")
# is equivalent with
account = Account("test", steem_instance=shared_steem_instance())
```

## beem.market

**class** `beem.market.Market` (`base=None`, `quote=None`, `steem_instance=None`)

Bases: dict

This class allows to easily access Markets on the blockchain for trading, etc.

### Parameters

- **steem\_instance** (`beem.steem.Steem`) – Steem instance
- **base** (`beem.asset.Asset`) – Base asset
- **quote** (`beem.asset.Asset`) – Quote asset

**Returns** Blockchain Market

**Return type** dictionary with overloaded methods

Instances of this class are dictionaries that come with additional methods (see below) that allow dealing with a market and its corresponding functions.

This class tries to identify **two** assets as provided in the parameters in one of the following forms:

- base and quote are valid assets (according to `beem.asset.Asset`)
- base:quote separated with :
- base/quote separated with /
- base-quote separated with -

---

**Note:** Throughout this library, the quote symbol will be presented first (e.g. STEEM:SBD with STEEM being the quote), while the base only refers to a secondary asset for a trade. This means, if you call `beem.market.Market.sell()` or `beem.market.Market.buy()`, you will sell/buy **only quote** and obtain/pay **only base**.

---

**accountopenorders** (`account=None`, `raw_data=False`)

Returns open Orders

### Parameters

- **account** (*steem.account.Account*) – Account name or instance of Account to show orders for in this market
- **raw\_data** (*bool*) – (optional) returns raw data if set True, or a list of Order() instances if False (defaults to False)

**static btc\_usd\_ticker** (*verbose=False*)

Returns the BTC/USD price from bitfinex, gdax, kraken, okcoin and bitstamp. The mean price is weighted by the exchange volume.

**buy** (*price, amount, expiration=None, killfill=False, account=None, orderid=None, returnOrderId=False*)  
Places a buy order in a given market

#### Parameters

- **price** (*float*) – price denoted in base/quote
- **amount** (*number*) – Amount of quote to buy
- **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** (*string*) – Account name that executes that order
- **returnOrderId** (*string*) – If set to “head” or “irreversible” the call will wait for the tx to appear in the head/irreversible block and add the key “orderid” to the tx output

Prices/Rates are denoted in ‘base’, i.e. the SBD\_STEEM market is priced in STEEM per SBD.

**Example:** in the SBD\_STEEM market, a price of 300 means a SBD is worth 300 STEEM

---

**Note:** All prices returned are in the **reversed** orientation as the market. I.e. in the STEEM/SBD market, prices are SBD per STEEM. That way you can multiply prices with *1.05* to get a +5%.

---

**Warning:** Since buy orders are placed as limit-sell orders for the base asset, you may end up obtaining more of the buy asset than you placed the order for. Example:

- You place an order to buy 10 SBD for 100 STEEM/SBD
- This means that you actually place a sell order for 1000 STEEM in order to obtain **at least** 10 SBD
- If an order on the market exists that sells SBD for cheaper, you will end up with more than 10 SBD

**cancel** (*orderNumbers, account=None, \*\*kwargs*)

Cancels an order you have placed in a given market. Requires only the “orderNumbers”.

**Parameters orderNumbers** (*int/list*) – A single order number or a list of order numbers

**get\_string** (*separator=':'*)

Return a formatted string that identifies the market, e.g. STEEM:SBD

**Parameters separator** (*str*) – The separator of the assets (defaults to :)

**market\_history** (*bucket\_seconds=300, start\_age=3600, end\_age=0, raw\_data=False*)

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)
- **raw\_data** (*bool*) – (optional) returns raw data if set True

**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
}
```

**market\_history\_buckets()****orderbook** (*limit=25, raw\_data=False*)

Returns the order book for SBD/STEEM market. :param int limit: Limit the amount of orders (default: 25)

**Sample output (raw\_data=False):**

```
{
  'asks': [
    380.510 STEEM 460.291 SBD @ 1.209669 SBD/STEEM,
    53.785 STEEM 65.063 SBD @ 1.209687 SBD/STEEM
  ],
  'bids': [
    0.292 STEEM 0.353 SBD @ 1.208904 SBD/STEEM,
    8.498 STEEM 10.262 SBD @ 1.207578 SBD/STEEM
  ],
  'asks_date': [
    datetime.datetime(2018, 4, 30, 21, 7, 24, tzinfo=<UTC>),
    datetime.datetime(2018, 4, 30, 18, 12, 18, tzinfo=<UTC>)
  ],
  'bids_date': [
    datetime.datetime(2018, 4, 30, 21, 1, 21, tzinfo=<UTC>),
    datetime.datetime(2018, 4, 30, 20, 38, 21, tzinfo=<UTC>)
  ]
}
```

**Sample output (raw\_data=True):**

```
{
  'asks': [
```

(continues on next page)

(continued from previous page)

```

    {
      'order_price': {'base': '8.000 STEEM', 'quote': '9.618 SBD'},
      'real_price': '1.20225000000000004',
      'steem': 4565,
      'sbd': 5488,
      'created': '2018-04-30T21:12:45'
    },
    'bids': [
      {
        'order_price': {'base': '10.000 SBD', 'quote': '8.333 STEEM'},
        'real_price': '1.20004800192007677',
        'steem': 8333,
        'sbd': 10000,
        'created': '2018-04-30T20:29:33'
      }
    ]
  }
}

```

**Note:** Each bid is an instance of class *beem.price.Order* and thus carries the keys *base*, *quote* and *price*. From those you can obtain the actual amounts for sale

#### **recent\_trades** (*limit=25, raw\_data=False*)

Returns the order book for a given market. You may also specify “all” to get the orderbooks of all markets.

##### **Parameters**

- **limit** (*int*) – Limit the amount of orders (default: 25)
- **raw\_data** (*bool*) – when False, FilledOrder objects will be returned

Sample output (*raw\_data=False*):

```

[
  (2018-04-30 21:00:54+00:00) 0.267 STEEM 0.323 SBD @ 1.209738 SBD/
↪STEEM,
  (2018-04-30 20:59:30+00:00) 0.131 STEEM 0.159 SBD @ 1.213740 SBD/
↪STEEM,
  (2018-04-30 20:55:45+00:00) 0.093 STEEM 0.113 SBD @ 1.215054 SBD/
↪STEEM,
  (2018-04-30 20:55:30+00:00) 26.501 STEEM 32.058 SBD @ 1.209690 SBD/
↪STEEM,
  (2018-04-30 20:55:18+00:00) 2.108 STEEM 2.550 SBD @ 1.209677 SBD/
↪STEEM,
]

```

Sample output (*raw\_data=True*):

```

[
  {'date': '2018-04-30T21:02:45', 'current_pays': '0.235 SBD', 'open_
↪pays': '0.194 STEEM'},
  {'date': '2018-04-30T21:02:03', 'current_pays': '24.494 SBD',
↪'open_pays': '20.248 STEEM'},
  {'date': '2018-04-30T20:48:30', 'current_pays': '175.464 STEEM',
↪'open_pays': '211.955 SBD'},
  {'date': '2018-04-30T20:48:30', 'current_pays': '0.999 STEEM',
↪'open_pays': '1.207 SBD'},
]

```

(continues on next page)

(continued from previous page)

```
{'date': '2018-04-30T20:47:54', 'current_pays': '0.273 SBD', 'open_
↪pays': '0.225 STEEM'},
]
```

**Note:** Each bid is an instance of class:*steem.price.Order* and thus carries the keys *base*, *quote* and *price*. From those you can obtain the actual amounts for sale

**sell** (*price*, *amount*, *expiration=None*, *killfill=False*, *account=None*, *orderid=None*, *returnOrderId=False*)  
Places a sell order in a given market

#### Parameters

- **price** (*float*) – price denoted in base/quote
- **amount** (*number*) – Amount of quote to sell
- **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** (*string*) – Account name that executes that order
- **returnOrderId** (*string*) – If set to “head” or “irreversible” the call will wait for the tx to appear in the head/irreversible block and add the key “orderid” to the tx output

Prices/Rates are denoted in ‘base’, i.e. the SBD\_STEEM market is priced in STEEM per SBD.

**Example:** in the SBD\_STEEM market, a price of 300 means a SBD is worth 300 STEEM

**Note:** All prices returned are in the **reversed** orientation as the market. I.e. in the STEEM/SBD market, prices are SBD per STEEM. That way you can multiply prices with *1.05* to get a +5%.

**static steem\_btc\_ticker()**

Returns the STEEM/BTC price from bittrex, binance, huobi and upbit. The mean price is weighted by the exchange volume.

**steem\_usd\_implied()**

Returns the current STEEM/USD market price

**ticker** (*raw\_data=False*)

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
}
```

**trade\_history** (*start=None, stop=None, intervall=None, limit=25, raw\_data=False*)

Returns the trade history for the internal market

This function allows to fetch a fixed number of trades at fixed intervall times to reduce the call duration time. E.g. it is possible to receive the trades from the last 7 days, by fetching 100 trades each 6 hours.

When intervall is set to None, all trades are received between start and stop. This can take a while.

#### Parameters

- **start** (*datetime*) – Start date
- **stop** (*datetime*) – Stop date
- **intervall** (*timedelta*) – Defines the intervall
- **limit** (*int*) – Defines how many trades are fetched at each intervall point
- **raw\_data** (*bool*) – when True, the raw data are returned

**trades** (*limit=100, start=None, stop=None, raw\_data=False*)

Returns your trade history for a given market.

#### Parameters

- **limit** (*int*) – Limit the amount of orders (default: 100)
- **start** (*datetime*) – start time
- **stop** (*datetime*) – stop time

**volume24h** (*raw\_data=False*)

Returns the 24-hour volume for all markets, plus totals for primary currencies.

Sample output:

```
{
  "STEEM": 361666.63617,
  "SBD": 1087.0
}
```

## beem.memo

**class** beem.memo.**Memo** (*from\_account=None, to\_account=None, steem\_instance=None*)

Bases: object

Deals with Memos that are attached to a transfer

### Parameters

- **from\_account** (`beem.account.Account`) – Account that has sent the memo
- **to\_account** (`beem.account.Account`) – Account that has received the memo
- **steem\_instance** (`beem.steem.Steem`) – Steem instance

A memo is encrypted with a shared secret derived from a private key of the sender and a public key of the receiver. Due to the underlying mathematics, the same shared secret can be derived by the private key of the receiver and the public key of the sender. The encrypted message is perturbed by a nonce that is part of the transmitted message.

```
from beem.memo import Memo
m = Memo("steemeu", "wallet.xeroc")
m.steem.wallet.unlock("secret")
enc = (m.encrypt("foobar"))
print(enc)
>> {'nonce': '17329630356955254641', 'message': '8563e2bb2976e0217806d642901a2855
↪ '}
print(m.decrypt(enc))
>> foobar
```

To decrypt a memo, simply use

```
from beem.memo import Memo
m = Memo()
m.steem.wallet.unlock("secret")
print(m.decrypt(op_data["memo"]))
```

if `op_data` being the payload of a transfer operation.

### Memo Keys

In Steem, memos are AES-256 encrypted with a shared secret between sender and receiver. It is derived from the memo private key of the sender and the memo public key of the receiver.

In order for the receiver to decode the memo, the shared secret has to be derived from the receiver's private key and the senders public key.

The memo public key is part of the account and can be retrieved with the `get_account` call:

```
get_account <accountname>
{
  [...]
  "options": {
    "memo_key": "GPH5TPTziKkLexhVKsQKtSpo4bAv5RnB8oXcG4sMHEwCcTf3r7dqE",
    [...]
  },
  [...]
}
```

while the memo private key can be dumped with `dump_private_keys`

### Memo Message

The take the following form:

```
{
  "from": "GPH5mgup8evDqMnT86L7scVebRYDC2fwAWmygPEUL43LjstQegYCC",
  "to": "GPH5Ar4j53kFWuEZQ9XhxbAja4YXMPJ2EnUg5QcrdeMFYUNMMNJbe",
```

(continues on next page)

(continued from previous page)

```
"nonce": "13043867485137706821",
"message": "d55524c37320920844ca83bb20c8d008"
}
```

The fields *from* and *to* contain the memo public key of sender and receiver. The *nonce* is a random integer that is used for the seed of the AES encryption of the message.

#### Encrypting a memo

The high level memo class makes use of the beem wallet to obtain keys for the corresponding accounts.

```
from beem.memo import Memo
from beem.account import Account

memoObj = Memo(
    from_account=Account(from_account),
    to_account=Account(to_account)
)
encrypted_memo = memoObj.encrypt(memo)
```

#### Decoding of a received memo

```
from getpass import getpass
from beem.block import Block
from beem.memo import Memo

# Obtain a transfer from the blockchain
block = Block(23755086)           # block
transaction = block["transactions"][3]  # transactions
op = transaction["operations"][0]      # operation
op_id = op[0]                     # operation type
op_data = op[1]                   # operation payload

# Instantiate Memo for decoding
memo = Memo()

# Unlock wallet
memo.unlock_wallet(getpass())

# Decode memo
# Raises exception if required keys not available in the wallet
print(memo.decrypt(op_data["transfer"]))
```

#### **decrypt** (memo)

Decrypt a memo

**Parameters** **memo** (*str*) – encrypted memo message

**Returns** encrypted memo

**Return type** str

#### **encrypt** (memo, *bts\_encrypt=False*)

Encrypt a memo

**Parameters** **memo** (*str*) – clear text memo message

**Returns** encrypted memo

**Return type** str

**unlock\_wallet** (\*args, \*\*kwargs)  
Unlock the library internal wallet

## beem.message

**class** beem.message.**Message** (message, steem\_instance=None)  
Bases: object

**sign** (account=None, \*\*kwargs)  
Sign a message with an account's memo key

**Parameters** **account** (str) – (optional) the account that owns the bet (defaults to default\_account)

**Returns** the signed message encapsulated in a known format

**verify** (\*\*kwargs)  
Verify a message with an account's memo key

**Parameters** **account** (str) – (optional) the account that owns the bet (defaults to default\_account)

**Returns** True if the message is verified successfully

**Raises** InvalidMessageSignature if the signature is not ok

## beem.notify

**class** beem.notify.**Notify** (on\_block=None, only\_block\_id=False, steem\_instance=None, keep\_alive=25)  
Bases: events.events.Events

Notifications on Blockchain events.

This modules allows you to be notified of events taking place on the blockchain.

### Parameters

- **on\_block** (fnt) – Callback that will be called for each block received
- **steem\_instance** (beem.steem.Steem) – Steem instance

### Example

```
from pprint import pprint
from beem.notify import Notify

notify = Notify(
    on_block=print,
)
notify.listen()
```

**close** ()  
Cleanly close the Notify instance

**listen** ()  
This call initiates the listening/notification process. It behaves similar to run\_forever().

**process\_block** (message)

**reset\_subscriptions** (accounts=[])  
Change the subscriptions of a running Notify instance

## beem.price

**class** `beem.price.FilledOrder` (*order*, *steem\_instance=None*, *\*\*kwargs*)

Bases: `beem.price.Price`

This class inherits `beem.price.Price` but has the `base` and `quote` Amounts not only be used to represent the price (as a ratio of base and quote) but instead has those amounts represent the amounts of an actually filled order!

**Parameters** `steem_instance` (`beem.steem.Steem`) – Steem instance

---

**Note:** Instances of this class come with an additional `date` key that shows when the order has been filled!

---

`json()`

**class** `beem.price.Order` (*base*, *quote=None*, *steem\_instance=None*, *\*\*kwargs*)

Bases: `beem.price.Price`

This class inherits `beem.price.Price` but has the `base` and `quote` Amounts not only be used to represent the price (as a ratio of base and quote) but instead has those amounts represent the amounts of an actual order!

**Parameters** `steem_instance` (`beem.steem.Steem`) – Steem instance

---

**Note:** If an order is marked as deleted, it will carry the ‘deleted’ key which is set to `True` and all other data be `None`.

---

**class** `beem.price.Price` (*price=None*, *base=None*, *quote=None*, *base\_asset=None*, *steem\_instance=None*)

Bases: `dict`

This class deals with all sorts of prices of any pair of assets to simplify dealing with the tuple:

`(quote, base)`

each being an instance of `beem.amount.Amount`. The amount themselves define the price.

---

**Note:** The price (floating) is derived as `base/quote`

---

### Parameters

- **args** (*list*) – Allows to deal with different representations of a price
- **base** (`beem.asset.Asset`) – Base asset
- **quote** (`beem.asset.Asset`) – Quote asset
- **steem\_instance** (`beem.steem.Steem`) – Steem instance

**Returns** All data required to represent a price

**Return type** `dict`

Way to obtain a proper instance:

- `args` is a str with a price and two assets
- `args` can be a floating number and `base` and `quote` being instances of `beem.asset.Asset`



- args can be a floating number and base and quote being instances of str
- args can be dict with keys price, base, and quote (*graphene balances*)
- args can be dict with keys base and quote
- args can be dict with key receives (filled orders)
- args being a list of [quote, base] both being instances of `beem.amount.Amount`
- args being a list of [quote, base] both being instances of str (amount symbol)
- base and quote being instances of `beem.asset.Amount`

This allows instantiations like:

- `Price("0.315 SBD/STEEM")`
- `Price(0.315, base="SBD", quote="STEEM")`
- `Price(0.315, base=Asset("SBD"), quote=Asset("STEEM"))`
- `Price({"base": {"amount": 1, "asset_id": "SBD"}, "quote": {"amount": 10, "asset_id": "SBD"}})`
- `Price(quote="10 STEEM", base="1 SBD")`
- `Price("10 STEEM", "1 SBD")`
- `Price(Amount("10 STEEM"), Amount("1 SBD"))`
- `Price(1.0, "SBD/STEEM")`

Instances of this class can be used in regular mathematical expressions (+-\*/%) such as:

```
>>> from beem.price import Price
>>> Price("0.3314 SBD/STEEM") * 2
0.662804 SBD/STEEM
>>> Price(0.3314, "SBD", "STEEM")
0.331402 SBD/STEEM
```

#### **as\_base** (base)

Returns the price instance so that the base asset is base.

Note: This makes a copy of the object!

```
>>> from beem.price import Price
>>> Price("0.3314 SBD/STEEM").as_base("STEEM")
3.017483 STEEM/SBD
```

#### **as\_quote** (quote)

Returns the price instance so that the quote asset is quote.

Note: This makes a copy of the object!

```
>>> from beem.price import Price
>>> Price("0.3314 SBD/STEEM").as_quote("SBD")
3.017483 STEEM/SBD
```

**copy** () → a shallow copy of D

#### **invert** ()

Invert the price (e.g. go from SBD/STEEM into STEEM/SBD)

```
>>> from beem.price import Price
>>> Price("0.3314 SBD/STEEM").invert()
3.017483 STEEM/SBD
```

**json()**

**market**

Open the corresponding market

**Returns** Instance of *beem.market.Market* for the corresponding pair of assets.

**symbols()**

## beem.rc

**class** *beem.rc.RC* (*steem\_instance=None*)

Bases: object

**account\_update\_dict** (*account\_update\_dict*)

Calc RC costs for account update

**claim\_account** (*tx\_size=300*)

Claim account

**comment** (*tx\_size=1000, permink\_length=10, parent\_permink\_length=10*)

Calc RC for a comment

**comment\_dict** (*comment\_dict*)

Calc RC costs for a comment dict object

Example for calculating RC costs

```
from beem.rc import RC
comment_dict = {
    "permlink": "test", "author": "holger80",
    "body": "test", "parent_permlink": "",
    "parent_author": "", "title": "test",
    "json_metadata": {"foo": "bar"}
}

rc = RC()
print(rc.comment_from_dict(comment_dict))
```

**custom\_json** (*tx\_size=444*)

**custom\_json\_dict** (*custom\_json\_dict*)

Calc RC costs for a custom\_json

Example for calculating RC costs

```
from beem.rc import RC
from collections import OrderedDict
custom_json_dict = {
    "json": [
        "reblog", OrderedDict([("account", "xeroc"), (
↵ "author", "chainsquad"),
        ("permlink", "streemian-
↵ com-to-open-its-doors-and-offer-a-20-discount")
    ])
]
```

(continues on next page)

(continued from previous page)

```

        },
        "required_auths": [],
        "required_posting_auths": ["xeroc"],
        "id": "follow"
    }

rc = RC()
print(rc.comment(custom_json_dict))

```

**get\_resource\_count** (*tx\_size*, *state\_bytes\_count=0*, *new\_account\_op\_count=0*, *market\_op\_count=0*)

Creates the resource\_count dictionary based on tx\_size, state\_bytes\_count, new\_account\_op\_count and market\_op\_count

**get\_tx\_size** (*op*)

Returns the tx size of an operation

**transfer** (*tx\_size=290*, *market\_op\_count=1*)

Calc RC of a transfer

**transfer\_dict** (*transfer\_dict*)

Calc RC costs for a transfer dict object

Example for calculating RC costs

```

from beem.rc import RC
from beem.amount import Amount
transfer_dict = {
    "from": "foo", "to": "baar",
    "amount": Amount("111.110 STEEM"),
    "memo": "Fooo"
}

rc = RC()
print(rc.comment(transfer_dict))

```

**vote** (*tx\_size=210*)

Calc RC for a vote

**vote\_dict** (*vote\_dict*)

Calc RC costs for a vote

Example for calculating RC costs

```

from beem.rc import RC
vote_dict = {
    "voter": "foobara", "author": "foobarc",
    "permlink": "foobard", "weight": 1000
}

rc = RC()
print(rc.comment(vote_dict))

```

## beem.snapshot

**class** beem.snapshot.**AccountSnapshot** (*account*, *account\_history=[]*, *steem\_instance=None*)

Bases: list

This class allows to easily access Account history

#### Parameters

- **account\_name** (*str*) – Name of the account
- **steem\_instance** (*beem.steem.Steem*) – Steem instance

**build** (*only\_ops=[]*, *exclude\_ops=[]*, *enable\_rewards=False*, *enable\_out\_votes=False*, *enable\_in\_votes=False*)

Builds the account history based on all account operations

#### Parameters

- **only\_ops** (*array*) – Limit generator by these operations (*optional*)
- **exclude\_ops** (*array*) – Exclude these operations from generator (*optional*)

**build\_curation\_arrays** (*end\_date=None*, *sum\_days=7*)

Build curation arrays

**build\_rep\_arrays** ()

Build reputation arrays

**build\_sp\_arrays** ()

Builds the own\_sp and eff\_sp array

**build\_vp\_arrays** ()

Build vote power arrays

**get\_account\_history** (*start=None*, *stop=None*, *use\_block\_num=True*)

Uses account history to fetch all related ops

#### Parameters

- **start** (*int/datetime*) – start number/date of transactions to return (*optional*)
- **stop** (*int/datetime*) – stop number/date of transactions to return (*optional*)
- **use\_block\_num** (*bool*) – if true, start and stop are block numbers, otherwise virtual OP count numbers.

**get\_data** (*timestamp=None*, *index=0*)

Returns snapshot for given timestamp

**get\_ops** (*start=None*, *stop=None*, *use\_block\_num=True*, *only\_ops=[]*, *exclude\_ops=[]*)

Returns ops in the given range

**parse\_op** (*op*, *only\_ops=[]*, *enable\_rewards=False*, *enable\_out\_votes=False*, *enable\_in\_votes=False*)

Parse account history operation

**reset** ()

Resets the arrays not the stored account history

**search** (*search\_str*, *start=None*, *stop=None*, *use\_block\_num=True*)

Returns ops in the given range

**update** (*timestamp*, *own*, *delegated\_in=None*, *delegated\_out=None*, *steem=0*, *sbd=0*)

Updates the internal state arrays

#### Parameters

- **timestamp** (*datetime*) – datetime of the update
- **own** (*Amount/float*) – vests
- **delegated\_in** (*dict*) – Incoming delegation

- **delegated\_out** (*dict*) – Outgoing delegation
- **steem** (*Amount/float*) – steem
- **sbd** (*Amount/float*) – sbd

**update\_in\_vote** (*timestamp, weight, op*)

**update\_out\_vote** (*timestamp, weight*)

**update\_rewards** (*timestamp, curation\_reward, author\_vests, author\_steem, author\_sbd*)

## beem.storage

**class** beem.storage.Configuration

Bases: *beem.storage.DataDir*

This is the configuration storage that stores key/value pairs in the *config* table of the SQLite3 database.

**checkBackup** ()

Backup the SQL database every 7 days

**config\_defaults** = {'client\_id': '', 'hot\_sign\_redirect\_uri': None, 'node': ['https

**create\_table** ()

Create the new table in the SQLite database

**delete** (*key*)

Delete a key from the configuration store

**exists\_table** ()

Check if the database table exists

**get** (*key, default=None*)

Return the key if exists or a default value

**items** ()

**nodelist** = [{'version': '0.20.2', 'score': 100, 'url': 'https://api.steemit.com', '}

Default configuration

**nodes** = ['https://api.steemit.com', 'https://steemd-appbase.steemit.com', 'wss://steem

**class** beem.storage.DataDir

Bases: *object*

This class ensures that the user's data is stored in its OS preotected user directory:

### OSX:

- *~/Library/Application Support/<AppName>*

### Windows:

- *C:\Documents and Settings<User>\Application Data\Local Settings<AppAuthor>\<AppName>*
- *C:\Documents and Settings<User>\Application Data<AppAuthor>\<AppName>*

### Linux:

- *~/local/share/<AppName>*

Furthermore, it offers an interface to generated backups in the *backups/* directory every now and then.

**appauthor** = 'beem'

```
appname = 'beem'

clean_data()
    Delete files older than 70 days

data_dir = '/home/docs/.local/share/beem'

mkdir_p()
    Ensure that the directory in which the data is stored exists

recover_with_latest_backup(backupdir='backups')
    Replace database with latest backup

refreshBackup()
    Make a new backup

sqlDataBaseFile = '/home/docs/.local/share/beem/beem.sqlite'

sqlite3_backup(backupdir)
    Create timestamped database copy

sqlite3_copy(src, dst)
    Copy sql file from src to dst

storageDatabase = 'beem.sqlite'

class beem.storage.Key
    Bases: beem.storage.DataDir

    This is the key storage that stores the public key and the (possibly encrypted) private key in the keys table in the
    SQLite3 database.

    add(wif, pub)

        Add a new public/private key pair (correspondence has to be checked elsewhere!)

        Parameters
        • pub(str) – Public key
        • wif(str) – Private key

    create_table()
        Create the new table in the SQLite database

    delete(pub)
        Delete the key identified as pub

        Parameters pub(str) – Public key

    exists_table()
        Check if the database table exists

    getPrivateKeyForPublicKey(pub)

        Returns the (possibly encrypted) private key that corresponds to a public key

        Parameters pub(str) – Public key

    The encryption scheme is BIP38

    getPublicKeys()
        Returns the public keys stored in the database
```

**updateWif** (*pub*, *wif*)

Change the wif to a pubkey

#### Parameters

- **pub** (*str*) – Public key
- **wif** (*str*) – Private key

**wipe** (*sure=False*)

Purge the entire wallet. No keys will survive this!

**class** `beem.storage.MasterPassword` (*password*)

Bases: `object`

The keys are encrypted with a Masterpassword that is stored in the configurationStore. It has a checksum to verify correctness of the password

**changePassword** (*newpassword*)

Change the password

**config\_key** = `'encrypted_master_password'`

This key identifies the encrypted master password stored in the confiration

**decryptEncryptedMaster** ()

Decrypt the encrypted masterpassword

**decrypted\_master** = `''`

**deriveChecksum** (*s*)

Derive the checksum

**getEncryptedMaster** ()

Obtain the encrypted masterkey

**newMaster** ()

Generate a new random masterpassword

**password** = `''`

**saveEncrytpedMaster** ()

Store the encrypted master password in the configuration store

**static wipe** (*sure=False*)

Remove all keys from configStorage

**class** `beem.storage.Token`

Bases: `beem.storage.DataDir`

This is the token storage that stores the public username and the (possibly encrypted) token in the *token* table in the SQLite3 database.

**add** (*name*, *token*)

Add a new public/private token pair (correspondence has to be checked elsewhere!)

#### Parameters

- **name** (*str*) – Public name
- **token** (*str*) – Private token

**create\_table** ()

Create the new table in the SQLite database

**delete** (*name*)

Delete the key identified as *name*

**Parameters** **name** (*str*) – Public name

**exists\_table** ()

Check if the database table exists

**getPublicNames** ()

Returns the public names stored in the database

**getTokenForPublicName** (*name*)

**Returns the (possibly encrypted) private token that** corresponds to a public name

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

The encryption scheme is BIP38

**updateToken** (*name*, *token*)

Change the token to a name

**Parameters**

- **name** (*str*) – Public name
- **token** (*str*) – Private token

**wipe** (*sure=False*)

Purge the entire wallet. No keys will survive this!

## beem.transactionbuilder

```
class beem.transactionbuilder.TransactionBuilder (tx={}, use_condenser_api=True,  
steem_instance=None, **kwargs)
```

Bases: dict

This class simplifies the creation of transactions by adding operations and signers. To build your own transactions and sign them

**Parameters**

- **tx** (*dict*) – transaction (Optional). If not set, the new transaction is created.
- **expiration** (*int*) – Delay in seconds until transactions are supposed to expire (*optional*) (default is 30)
- **steem\_instance** (*Steem*) – If not set, `shared_steem_instance()` is used

```
from beem.transactionbuilder import TransactionBuilder
from beembase.operations import Transfer
from beem import Steem
wif = "5KQwrPbwdL6PhXujxW37FSSQZ1JiwsST4cqQzDeyXtP79zkvFD3"
stm = Steem(nobroadcast=True, keys={'active': wif})
tx = TransactionBuilder(steem_instance=stm)
transfer = {"from": "test", "to": "test1", "amount": "1 STEEM", "memo": ""}
tx.appendOps(Transfer(transfer))
tx.appendSigner("test", "active") # or tx.appendWif(wif)
signed_tx = tx.sign()
broadcast_tx = tx.broadcast()
```



**addSigningInformation** (*account, permission, reconstruct\_tx=False*)

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)

Not needed when “appendWif” was already or is going to be used

FIXME: Does not work with owner keys!

**Parameters** **reconstruct\_tx** (*bool*) – when set to False and tx is already constructed, it will not be reconstructed and already added signatures remain

**appendMissingSignatures** ()

Store which accounts/keys are supposed to sign the transaction

This method is used for an offline-signer!

**appendOps** (*ops, append\_to=None*)

Append op(s) to the transaction builder

**Parameters** **ops** (*list*) – One or a list of operations

**appendSigner** (*account, permission*)

Try to obtain the wif key from the wallet by telling which account and permission is supposed to sign the transaction. It is possible to add more than one signer.

**appendWif** (*wif*)

Add a wif that should be used for signing of the transaction.

**Parameters** **wif** (*string*) – One wif key to use for signing a transaction.

**broadcast** (*max\_block\_age=-1*)

Broadcast a transaction to the steem network. Returns the signed transaction and clears itself after broadcast.

Clears itself when broadcast was not successful.

**Parameters** **max\_block\_age** (*int*) – parameter only used for appbase ready nodes

**clear** ()

Clear the transaction builder and start from scratch

**clearWifs** ()

Clear all stored wifs

**constructTx** (*ref\_block\_num=None, ref\_block\_prefix=None*)

Construct the actual transaction and store it in the class’s dict store

**get\_parent** ()

TransactionBuilders don’t have parents, they are their own parent

**get\_potential\_signatures** ()

Returns public key from signature

**get\_required\_signatures** (*available\_keys=[]*)

Returns public key from signature

**get\_transaction\_hex** ()

Returns a hex value of the transaction

**is\_empty** ()

Check if ops is empty

**json** ()

Show the transaction as plain json

**list\_operations()**

List all ops

**set\_expiration(p)**

Set expiration date

**sign(reconstruct\_tx=True)**

Sign a provided transaction with the provided key(s) One or many wif keys to use for signing a transaction. The wif keys can be provided by “appendWif” or the signer can be defined “appendSigner”. The wif keys from all signer that are defined by “appendSigner will be loaded from the wallet.

**Parameters reconstruct\_tx (bool)** – when set to False and tx is already constructed, it will not be reconstructed and already added signatures remain

**verify\_authority()**

Verify the authority of the signed transaction

## beem.utils

**beem.utils.addTzInfo(t, timezone='UTC')**

Returns a datetime object with tzinfo added

**beem.utils.assets\_from\_string(text)**

Correctly split a string containing an asset pair.

Splits the string into two assets with the separator being one of the following: :, /, or -.

**beem.utils.construct\_authorperm(\*args)**

Create a post identifier from comment/post object or arguments. Examples:

```
>>> from beem.utils import construct_authorperm
>>> print(construct_authorperm('username', 'permlink'))
@username/permlink
>>> print(construct_authorperm({'author': 'username', 'permlink':
↳ 'permlink'}))
@username/permlink
```

**beem.utils.construct\_authorpermvoter(\*args)**

Create a vote identifier from vote object or arguments. Examples:

```
>>> from beem.utils import construct_authorpermvoter
>>> print(construct_authorpermvoter('username', 'permlink', 'voter'))
@username/permlink|voter
>>> print(construct_authorpermvoter({'author': 'username', 'permlink':
↳ 'permlink', 'voter': 'voter'}))
@username/permlink|voter
```

**beem.utils.derive\_permlink(title, parent\_permlink=None, parent\_author=None)**

**beem.utils.findall\_patch\_hunks(body=None)**

**beem.utils.formatTime(t)**

Properly Format Time for permlinks

**beem.utils.formatTimeFromNow(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 formatted time for Graphene (%Y-%m-%dT%H:%M:%S)

**Return type** str

`beem.utils.formatTimeString(t)`  
Properly Format Time for permlinks

`beem.utils.formatTimedelta(td)`  
Format timedelta to String

`beem.utils.formatToTimeStamp(t)`  
Returns a timestamp integer

**Parameters** *t* (*datetime*) – datetime object

**Returns** Timestamp as integer

`beem.utils.make_patch(a, b, n=3)`

`beem.utils.parse_time(block_time)`

Take a string representation of time from the blockchain, and parse it into datetime object.

`beem.utils.remove_from_dict(obj, keys=[], keep_keys=True)`  
Prune a class or dictionary of all but keys (keep\_keys=True). Prune a class or dictionary of specified keys.(keep\_keys=False).

`beem.utils.reputation_to_score(rep)`  
Converts the account reputation value into the reputation score

`beem.utils.resolve_authorperm(identifier)`  
Correctly split a string containing an authorperm.

Splits the string into author and permlink with the following separator: /.

Examples:

```
>>> from beem.utils import resolve_authorperm
>>> author, permlink = resolve_authorperm('https://d.tube/#!/v/pottlund/
↳m5cqkd1a')
>>> author, permlink = resolve_authorperm("https://steemit.com/witness-
↳category/@gtg/24lfrm-gtg-witness-log")
>>> author, permlink = resolve_authorperm("@gtg/24lfrm-gtg-witness-log")
>>> author, permlink = resolve_authorperm("https://busy.org/@gtg/24lfrm-
↳gtg-witness-log")
```

`beem.utils.resolve_authorpermvoter(identifier)`  
Correctly split a string containing an authorpermvoter.

Splits the string into author and permlink with the following separator: / and |.

`beem.utils.resolve_root_identifier(url)`

`beem.utils.sanitize_permlink(permlink)`

## beem.vote

**class** `beem.vote.AccountVotes(account, start=None, stop=None, lazy=False, full=False, steem_instance=None)`

Bases: `beem.vote.VotesObject`

Obtain a list of votes for an account Lists the last 100+ votes on the given account.

**Parameters**

- **account** (*str*) – Account name

- **steem\_instance** (*steem*) – Steem() instance to use when accessing a RPC

**class** beem.vote.ActiveVotes (*authorperm, lazy=False, full=False, steem\_instance=None*)

Bases: *beem.vote.VotesObject*

Obtain a list of votes for a post

#### Parameters

- **authorperm** (*str*) – authorperm link
- **steem\_instance** (*steem*) – Steem() instance to use when accessing a RPC

**class** beem.vote.Vote (*voter, authorperm=None, full=False, lazy=False, steem\_instance=None*)

Bases: *beem.blockchainobject.BlockchainObject*

Read data about a Vote in the chain

#### Parameters

- **authorperm** (*str*) – perm link to post/comment
- **steem\_instance** (*steem*) – Steem() instance to use when accessing a RPC

```
>>> from beem.vote import Vote
>>> v = Vote("@gtg/steem-pressure-4-need-for-speed|gandalf")
```

**authorperm**

**json()**

**percent**

**refresh()**

**rep**

**reputation**

**rshares**

**sbd**

**time**

**type\_id = 11**

**votee**

**voter**

**weight**

**class** beem.vote.VotesObject

Bases: *list*

**get\_list** (*var='voter', voter=None, votee=None, start=None, stop=None, start\_percent=None, stop\_percent=None, sort\_key='time', reverse=True*)

**get\_sorted\_list** (*sort\_key='time', reverse=True*)

**printAsTable** (*voter=None, votee=None, start=None, stop=None, start\_percent=None, stop\_percent=None, sort\_key='time', reverse=True, allow\_refresh=True, return\_str=False, \*\*kwargs*)

**print\_stats** (*return\_str=False, \*\*kwargs*)

## beem.wallet

```
class beem.wallet.Wallet (steem_instance=None, *args, **kwargs)
```

Bases: object

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** (`SteemNodeRPC`) – 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, beem 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!

A new wallet can be created by using:

```
from beem import Steem
steem = Steem()
steem.wallet.wipe(True)
steem.wallet.create("supersecret-passphrase")
```

This will raise an exception if you already have a wallet installed.

The wallet can be unlocked for signing using

```
from beem import Steem
steem = Steem()
steem.wallet.unlock("supersecret-passphrase")
```

A private key can be added by using the `steem.wallet.Wallet.addPrivateKey()` method that is available **after** unlocking the wallet with the correct passphrase:

```
from beem import Steem
steem = Steem()
steem.wallet.unlock("supersecret-passphrase")
steem.wallet.addPrivateKey("5xxxxxxxxxxxxxxxxxxxxx")
```

**Note:** The private key has to be either in hexadecimal or in wallet import format (wif) (starting with a 5).

**MasterPassword = None**

**addPrivateKey** (*wif*)

Add a private key to the wallet database

**Parameters** **wif** (*str*) – Private key

**addToken** (*name, token*)

**changePassphrase** (*new\_pwd*)

Change the passphrase for the wallet database

**clear\_local\_keys** ()

Clear all manually provided keys

**clear\_local\_token** ()

Clear all manually provided token

**configStorage** = **None**

**create** (*pwd*)

Alias for newWallet()

**created** ()

Do we have a wallet database already?

**decrypt\_token** (*enctoken*)

decrypt a wif key

**decrypt\_wif** (*encwif*)

decrypt a wif key

**deriveChecksum** (*s*)

Derive the checksum

**encrypt\_token** (*token*)

Encrypt a token key

**encrypt\_wif** (*wif*)

Encrypt a wif key

**getAccount** (*pub*)

Get the account data for a public key (first account found for this public key)

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

**getAccountFromPrivateKey** (*wif*)

Obtain account name from private key

**getAccountFromPublicKey** (*pub*)

Obtain the first account name from public key

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

Note: this returns only the first account with the given key. To get all accounts associated with a given public key, use `getAccountsFromPublicKey`.

**getAccounts** ()

Return all accounts installed in the wallet database

**getAccountsFromPublicKey** (*pub*)

Obtain all account names associated with a public key

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

**getActiveKeyForAccount** (*name*)

Obtain owner Active Key for an account from the wallet database

**getActiveKeysForAccount** (*name*)

Obtain list of all owner Active Keys for an account from the wallet database

**getAllAccounts** (*pub*)

Get the account data for a public key (all accounts found for this public key)

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

**getKeyForAccount** (*name*, *key\_type*)

Obtain *key\_type* Private Key for an account from the wallet database

**Parameters**

- **name** (*str*) – Account name
- **key\_type** (*str*) – key type, has to be one of “owner”, “active”, “posting” or “memo”

**getKeyType** (*account*, *pub*)

Get key type

**Parameters**

- **account** (*beem.account.Account/dict*) – Account data
- **pub** (*str*) – Public key

**getKeysForAccount** (*name*, *key\_type*)

Obtain a List of *key\_type* Private Keys for an account from the wallet database

**Parameters**

- **name** (*str*) – Account name
- **key\_type** (*str*) – key type, has to be one of “owner”, “active”, “posting” or “memo”

**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

**getOwnerKeysForAccount** (*name*)

Obtain list of all owner Private Keys for an account from the wallet database

**getPostingKeyForAccount** (*name*)

Obtain owner Posting Key for an account from the wallet database

**getPostingKeysForAccount** (*name*)

Obtain list of all owner Posting Keys 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

**getPublicNames** ()

Return all installed public token

**getTokenForAccountName** (*name*)

Obtain the private token for a given public name

**Parameters** **name** (*str*) – Public name

**keyMap** = {}

**keyStorage** = None

**keys** = {}

**lock()**  
Lock the wallet database

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

**masterpassword = None**

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

**prefix**

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

**Parameters** **account** (*str*) – name of account to be removed

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

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

**removeTokenFromPublicKeyName(name)**  
Remove a token from the wallet database

**Parameters** **name** (*str*) – token to be removed

**rpc**

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

**setToken(loadtoken)**  
This method is strictly only for in memory token that are passed to Wallet/Steem with the `token` argument

**token = {}**

**tokenStorage = None**

**tryUnlockFromEnv()**  
Try to fetch the unlock password from UNLOCK environment variable and keyring when no password is given.

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

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

**wipe(sure=False)**  
Purge all data in wallet database

## beem.witness

**class** `beem.witness.ListWitnesses` (*from\_account=""*, *limit=100*, *lazy=False*, *full=False*,  
*steem\_instance=None*)  
Bases: `beem.witness.WitnessesObject`  
List witnesses ranked by name

**Parameters**

- **from\_account** (*str*) – Witness name from which the lists starts (default = "")



- **limit** (*int*) – Limits the number of shown witnesses (default = 100)
- **steem\_instance** (*steem*) – Steem() instance to use when accessing a RPC

```
>>> from beem.witness import ListWitnesses
>>> ListWitnesses(from_account="gtg", limit=100)
<ListWitnesses gtg>
```

**class** beem.witness.**Witness** (*owner, full=False, lazy=False, steem\_instance=None*)

Bases: *beem.blockchainobject.BlockchainObject*

Read data about a witness in the chain

#### Parameters

- **account\_name** (*str*) – Name of the witness
- **steem\_instance** (*steem*) – Steem() instance to use when accessing a RPC

```
>>> from beem.witness import Witness
>>> Witness("gtg")
<Witness gtg>
```

#### account

**feed\_publish** (*base, quote=None, account=None*)

Publish a feed price as a witness. :param float base: USD Price of STEEM in SBD (implied price) :param float quote: (optional) Quote Price. Should be 1.000 (default), unless we are adjusting the feed to support the peg. :param str account: (optional) the source account for the transfer if not self["owner"]

#### is\_active

**json**()

**refresh**()

**type\_id** = 3

**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,
}
```

**class** beem.witness.**Witnesses** (*lazy=False, full=True, steem\_instance=None*)

Bases: *beem.witness.WitnessesObject*

Obtain a list of **active** witnesses and the current schedule

**Parameters** **steem\_instance** (*steem*) – Steem() instance to use when accessing a RPC

```
>>> from beem.witness import Witnesses
>>> Witnesses()
<Witnesses >
```

**refresh()**

**class** beem.witness.WitnessesObject

Bases: list

**get\_votes\_sum()**

**printAsTable** (sort\_key='votes', reverse=True, return\_str=False, \*\*kwargs)

**class** beem.witness.WitnessesRankedByVote (from\_account="", limit=100, lazy=False, full=False, steem\_instance=None)

Bases: *beem.witness.WitnessesObject*

Obtain a list of witnesses ranked by Vote

#### Parameters

- **from\_account** (*str*) – Witness name from which the lists starts (default = "")
- **limit** (*int*) – Limits the number of shown witnesses (default = 100)
- **steem\_instance** (*steem*) – Steem() instance to use when accessing a RPC

```
>>> from beem.witness import WitnessesRankedByVote
>>> WitnessesRankedByVote(limit=100)
<WitnessesRankedByVote >
```

**class** beem.witness.WitnessesVotedByAccount (account, lazy=False, full=True, steem\_instance=None)

Bases: *beem.witness.WitnessesObject*

Obtain a list of witnesses which have been voted by an account

#### Parameters

- **account** (*str*) – Account name
- **steem\_instance** (*steem*) – Steem() instance to use when accessing a RPC

```
>>> from beem.witness import WitnessesVotedByAccount
>>> WitnessesVotedByAccount("gtg")
<WitnessesVotedByAccount gtg>
```

## 3.7.2 beemapi Modules

### beemapi.steemnodeRPC

**class** beemapi.steemnodeRPC.SteemNodeRPC (\*args, \*\*kwargs)

Bases: *beemapi.graphenerpc.GrapheneRPC*

This class allows to call API methods exposed by the witness node via websockets / rpc-json.

#### Parameters

- **urls** (*str*) – Either a single Websocket/Http URL, or a list of URLs
- **user** (*str*) – Username for Authentication

- **password** (*str*) – Password for Authentication
- **num\_retries** (*int*) – Try x times to num\_retries to a node on disconnect, -1 for indefinitely
- **num\_retries\_call** (*int*) – Repeat num\_retries\_call times a rpc call on node error (default is 5)
- **timeout** (*int*) – Timeout setting for https nodes (default is 60)
- **use\_condenser** (*bool*) – Use the old condenser\_api rpc protocol on nodes with version 0.19.4 or higher. The settings has no effect on nodes with version of 0.19.3 or lower.

**get\_account** (*name*, *\*\*kwargs*)

Get full account details from account name

**Parameters** **name** (*str*) – Account name

**rpcexec** (*payload*)

Execute a call by sending the payload. It makes use of the GrapheneRPC library. In here, we mostly deal with Steem specific error handling

**Parameters** **payload** (*json*) – Payload data

**Raises**

- **ValueError** – if the server does not respond in proper JSON format
- **RPCError** – if the server returns an error

**set\_next\_node\_on\_empty\_reply** (*next\_node\_on\_empty\_reply=True*)

Switch to next node on empty reply for the next rpc call

## beemapi.exceptions

**exception** beemapi.exceptions.**ApiNotSupported**

Bases: *beemapi.exceptions.RPCError*

**exception** beemapi.exceptions.**CallRetriesReached**

Bases: Exception

CallRetriesReached Exception. Only for internal use

**exception** beemapi.exceptions.**InvalidEndpointUrl**

Bases: Exception

**exception** beemapi.exceptions.**MissingRequiredActiveAuthority**

Bases: *beemapi.exceptions.RPCError*

**exception** beemapi.exceptions.**NoAccessApi**

Bases: *beemapi.exceptions.RPCError*

**exception** beemapi.exceptions.**NoApiWithName**

Bases: *beemapi.exceptions.RPCError*

**exception** beemapi.exceptions.**NoMethodWithName**

Bases: *beemapi.exceptions.RPCError*

**exception** beemapi.exceptions.**NumRetriesReached**

Bases: Exception

NumRetriesReached Exception.

**exception** `beemapi.exceptions.RPCConnection`

Bases: `Exception`

`RPCConnection` Exception.

**exception** `beemapi.exceptions.RPCError`

Bases: `Exception`

`RPCError` Exception.

**exception** `beemapi.exceptions.RPCErrorDoRetry`

Bases: `Exception`

`RPCErrorDoRetry` Exception.

**exception** `beemapi.exceptions.TimeoutException`

Bases: `Exception`

**exception** `beemapi.exceptions.UnauthorizedError`

Bases: `Exception`

`UnauthorizedError` Exception.

**exception** `beemapi.exceptions.UnhandledRPCError`

Bases: `beemapi.exceptions.RPCError`

**exception** `beemapi.exceptions.UnkownKey`

Bases: `beemapi.exceptions.RPCError`

**exception** `beemapi.exceptions.UnnecessarySignatureDetected`

Bases: `Exception`

**exception** `beemapi.exceptions.WorkingNodeMissing`

Bases: `Exception`

`beemapi.exceptions.decodeRPCErrorMsg(e)`

Helper function to decode the raised Exception and give it a python Exception class

## beemapi.websocket

This class allows subscribe to push notifications from the Steem node.

```
from pprint import pprint
from beemapi.websocket import SteemWebsocket

ws = SteemWebsocket(
    "wss://gtg.steem.house:8090",
    accounts=["test"],
    on_block=print,
)

ws.run_forever()
```

**class** `beemapi.websocket.SteemWebsocket` (*urls*, *user*="", *password*="", *only\_block\_id*=False, *on\_block*=None, *keep\_alive*=25, *num\_retries*=-1, *timeout*=60, *\*args*, *\*\*kwargs*)

Create a websocket connection and request push notifications

### Parameters

- **urls** (*str*) – Either a single Websocket URL, or a list of URLs

- **user** (*str*) – Username for Authentication
- **password** (*str*) – Password for Authentication
- **keep\_alive** (*int*) – seconds between a ping to the backend (defaults to 25seconds)

After instantiating this class, you can add event slots for:

- `on_block`

which will be called accordingly with the notification message received from the Steem node:

```
ws = SteemWebsocket (
    "wss://gtg.steem.house:8090",
)
ws.on_block += print
ws.run_forever()
```

**\_\_SteemWebsocket\_\_set\_subscriptions()**

set subscriptions ot on\_block function

**\_\_events\_\_** = ['on\_block']

**\_\_getattr\_\_** (*name*)

Map all methods to RPC calls and pass through the arguments

**\_\_init\_\_** (*urls*, *user*=", *password*=", *only\_block\_id*=False, *on\_block*=None, *keep\_alive*=25, *num\_retries*=-1, *timeout*=60, *\*args*, *\*\*kwargs*)

Initialize self. See help(type(self)) for accurate signature.

**\_\_module\_\_** = 'beemapi.websocket'

**\_ping** ()

Send keep\_alive request

**cancel\_subscriptions** ()

cancel\_all\_subscriptions removed from api

**close** ()

Closes the websocket connection and waits for the ping thread to close

**get\_request\_id** ()

Generates next request id

**on\_close** (*ws*)

Called when websocket connection is closed

**on\_error** (*ws*, *error*)

Called on websocket errors

**on\_message** (*ws*, *reply*, *\*args*)

This method is called by the websocket connection on every message that is received. If we receive a notice, we hand over post-processing and signalling of events to `process_notice`.

**on\_open** (*ws*)

This method will be called once the websocket connection is established. It will

- login,
- register to the database api, and
- subscribe to the objects defined if there is a callback/slot available for callbacks

**process\_block** (*data*)

This method is called on notices that need processing. Here, we call the `on_block` slot.

**reset\_subscriptions** (*accounts=[]*)

Reset subscriptions

**rpcexec** (*payload*)

Execute a call by sending the payload.

**Parameters** **payload** (*json*) – Payload data

**Raises**

- **ValueError** – if the server does not respond in proper JSON format
- **RPCError** – if the server returns an error

**run\_forever** ()

This method is used to run the websocket app continuously. It will execute callbacks as defined and try to stay connected with the provided APIs

**stop** ()

Stop running Websocket

## beemapi.node

**class** beemapi.node.**Node** (*url*)

Bases: object

**class** beemapi.node.**Nodes** (*urls, num\_retries, num\_retries\_call*)

Bases: list

Stores Node URLs and error counts

**error\_cnt**

**error\_cnt\_call**

**export\_working\_nodes** ()

**increase\_error\_cnt** ()

Increase node error count for current node

**increase\_error\_cnt\_call** ()

Increase call error count for current node

**next** ()

**node**

**num\_retries\_call\_reached**

**reset\_error\_cnt** ()

Set node error count for current node to zero

**reset\_error\_cnt\_call** ()

Set call error count for current node to zero

**sleep\_and\_check\_retries** (*errorMsg=None, sleep=True, call\_retry=False, showMsg=True*)

Sleep and check if num\_retries is reached

**url**

**working\_nodes\_count**

## beemapi.graphenerpc

---

**Note:** This is a low level class that can be used in combination with GrapheneClient

---

This class allows to call API methods exposed by the witness node via websockets. It does **not** support notifications and is not run asynchronously.

graphenewsrpc.

**class** beemapi.graphenerpc.GrapheneRPC (*urls*, *user=None*, *password=None*, *\*\*kwargs*)

Bases: object

This class allows to call API methods synchronously, without callbacks.

It logs warnings and errors.

### Parameters

- **urls** (*str*) – Either a single Websocket/Http URL, or a list of URLs
- **user** (*str*) – Username for Authentication
- **password** (*str*) – Password for Authentication
- **num\_retries** (*int*) – Try x times to num\_retries to a node on disconnect, -1 for indefinitely
- **num\_retries\_call** (*int*) – Repeat num\_retries\_call times a rpc call on node error (default is 5)
- **timeout** (*int*) – Timeout setting for https nodes (default is 60)
- **autoconnect** (*bool*) – When set to false, connection is performed on the first rpc call (default is True)
- **use\_condenser** (*bool*) – Use the old condenser\_api rpc protocol on nodes with version 0.19.4 or higher. The settings has no effect on nodes with version of 0.19.3 or lower.
- **custom\_chains** (*dict*) – custom chain which should be added to the known chains

Available APIs:

- database
- network\_node
- network\_broadcast

Usage:

```
from beemapi.graphenerpc import GrapheneRPC
ws = GrapheneRPC("wss://steemd.pevo.science", "", "")
print(ws.get_account_count())

ws = GrapheneRPC("https://api.steemit.com", "", "")
print(ws.get_account_count())
```

---

**Note:** This class allows to call methods available via websocket. If you want to use the notification subsystem, please use GrapheneWebsocket instead.

---

**error\_cnt**

**error\_cnt\_call**

**get\_network** (*props=None*)

Identify the connected network. This call returns a dictionary with keys chain\_id, core\_symbol and prefix

**get\_request\_id** ()

Get request id.

**get\_use\_appbase** ()

Returns True if appbase ready and appbase calls are set

**is\_appbase\_ready** ()

Check if node is appbase ready

**next** ()

Switches to the next node url

**num\_retries**

**num\_retries\_call**

**request\_send** (*payload*)

**rpcclose** ()

Close Websocket

**rpconnect** (*next\_url=True*)

Connect to next url in a loop.

**rpcexec** (*payload*)

Execute a call by sending the payload.

**Parameters** **payload** (*json*) – Payload data

**Raises**

- **ValueError** – if the server does not respond in proper JSON format
- **RPCEError** – if the server returns an error

**rpclogin** (*user, password*)

Login into Websocket

**version\_string\_to\_int** (*network\_version*)

**ws\_send** (*payload*)

**class** beemapi.graphenerpc.**SessionInstance**

Bases: object

Singelton for the Session Instance

**instance** = None

beemapi.graphenerpc.**create\_ws\_instance** (*use\_ssl=True, enable\_multithread=True*)

Get websocket instance

beemapi.graphenerpc.**set\_session\_instance** (*instance*)

Set session instance

beemapi.graphenerpc.**shared\_session\_instance** ()

Get session instance



### 3.7.3 beembase Modules

#### beembase.memo

`beembase.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

`beembase.memo.decode_memo_bts(priv, pub, nonce, message)`

Decode a message with a shared secret between Alice and Bob

##### Parameters

- **priv** (`PrivateKey`) – Private Key (of Bob)
- **pub** (`PublicKey`) – Public Key (of Alice)
- **nonce** (`int`) – Nonce used for Encryption
- **message** (`bytes`) – Encrypted Memo message

**Returns** Decrypted message

**Return type** `str`

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

`beembase.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`

`beembase.memo.encode_memo_bts(priv, pub, nonce, message)`

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

`beembase.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:

$$\text{Pub}(\text{Alice}) * \text{Priv}(\text{Bob}) = \text{Pub}(\text{Bob}) * \text{Priv}(\text{Alice})$$

`beembase.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

`beembase.memo.init_aes_bts(shared_secret, nonce)`

Initialize AES instance

**Parameters**

- **shared\_secret** (`hex`) – Shared Secret to use as encryption key
- **nonce** (`int`) – Random nonce

**Returns** AES instance

**Return type** AES

## beembase.objects

**class** `beembase.objects.Amount(d, prefix='STM')`

Bases: `object`

**class** `beembase.objects.Beneficiaries(*args, **kwargs)`

Bases: `beemgraphenebase.objects.GrapheneObject`

**class** `beembase.objects.Beneficiary(*args, **kwargs)`

Bases: `beemgraphenebase.objects.GrapheneObject`

**class** `beembase.objects.CommentOptionExtensions(o)`

Bases: `beemgraphenebase.types.Static_variant`

Serialize Comment Payout Beneficiaries.

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

Example:

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

```
class beembase.objects.ExchangeRate(*args,**kwargs)
    Bases: beemgraphenebase.objects.GrapheneObject
```

```
class beembase.objects.Extension(d)
    Bases: beemgraphenebase.types.Array
```

```
class beembase.objects.Memo (*args, **kwargs)
    Bases: beemgraphenebase.objects.GrapheneObject
```

```
class beembase.objects.Operation(*args, **kwargs)
    Bases: beemgraphenebase.objects.Operation
```

**getOperationNameForId**(*i*)  
Convert an operation id into the corresponding string

**json()**

```
operations ()
```

```
class beembase.objects.Permission(*args, **kwargs)
    Bases: beemgraphenebase.objects.GrapheneObject
```

```
class beembase.objects.Price(*args, **kwargs)
    Bases: beemgraphenebase.objects.GrapheneObject
```

```
class beembase.objects.WitnessProps(*args,**kwargs)
    Bases: beemgraphenebase.objects.GrapheneObject
```

## beembase.objecttypes

```
beembase.objecttypes.object_type = {'account':    2, 'account_history': 18, 'block_summary'
```

Object types for object ids

## beembase.operationids

```
beembase.operationids.getOperationNameForId(i)
```

Convert an operation id into the corresponding string

```
beembase.operationids.ops = ['vote', 'comment', 'transfer', 'transfer_to_vesting', 'withdr
Operation ids
```

## beembase.operations

```
beembase.operationids.getOperationNameForId(i)
```

Convert an operation id into the corresponding string

```
beembase.operationids.ops = ['vote', 'comment', 'transfer', 'transfer_to_vesting', 'withdr
Operation ids
```

## beembase.signedtransactions

**class** beembase.signedtransactions.**Signed\_Transaction** (\*args, \*\*kwargs)

Bases: *beemgraphenebase.signedtransactions.Signed\_Transaction*

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
- **custom\_chains** (*dict*) – custom chain which should be added to the known chains

**add\_custom\_chains** (*custom\_chain*)

**getKnownChains** ()

**getOperationKlass** ()

**sign** (*wifkeys*, *chain*='STEEM')

Sign the transaction with the provided private keys.

### Parameters

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

**verify** (*pubkeys*=[], *chain*='STEEM', *recover\_parameter*=False)

Returned pubkeys have to be checked if they are existing

## beembase.transactions

beembase.transactions.**getBlockParams** (*ws*)

Auxiliary method to obtain ref\_block\_num and ref\_block\_prefix. Requires a websocket connection to a witness node!

## 3.7.4 beemgraphenebase Modules

### beemgraphenebase.account

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

Bases: object

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 STM)

Example:

```
Address("STMFN9r6VYzBK8EKtMewfNbfiGCr56pHDBFi")
```

**derive256address\_with\_version** (*version=56*)

Derive address using RIPEMD160 (SHA256 (x) ) and adding version + checksum

**derivesha256address** ()

Derive address using RIPEMD160 (SHA256 (x) )

**derivesha512address** ()

Derive address using RIPEMD160 (SHA512 (x) )

**get\_public\_key** ()

Returns the pubkey

**class** beemgraphenebase.account.**BrainKey** (*brainkey=None, sequence=0*)

Bases: object

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\_blind\_private** ()

Derive private key from the brain key (and no sequence number)

**get\_brainkey** ()

Return brain key of this instance

**get\_private** ()

Derive private key from the brain key and the current sequence number

**get\_private\_key** ()

**get\_public** ()

**get\_public\_key** ()

**next\_sequence** ()

Increment the sequence number by 1

**normalize** (*brainkey*)

Correct formatting 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** beemgraphenebase.account.**PasswordKey** (*account, password, role='active', prefix='STM'*)

Bases: object

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

**get\_private\_key()**

**get\_public()**

**get\_public\_key()**

**class** beemgraphenebase.account.**PrivateKey** (*wif=None, prefix='STM'*)

Bases: *beemgraphenebase.account.PublicKey*

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 STM)

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.

**child** (*offset256*)

Derive new private key from this key and a sha256 “offset”

**compressedpubkey()**

Derive uncompressed public key

**derive\_from\_seed** (*offset*)

Derive private key using “generate\_from\_seed” method. Here, the key itself serves as a *seed*, and *offset* is expected to be a sha256 digest.

**derive\_private\_key** (*sequence*)

Derive new private key from this private key and an arbitrary sequence number

**get\_public\_key()**

Returns the pubkey

**get\_secret()**

Get sha256 digest of the wif key.

**class** beemgraphenebase.account.**PublicKey** (*pk, prefix='STM'*)

Bases: *beemgraphenebase.account.Address*

This class deals with Public Keys and inherits *Address*.

**Parameters**

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

Example::

```
PublicKey("STM6UtYWWs3rkZGV8JA86qrgkG6tyFksgECefKE1MiH4HkLD8PFGL")
```

**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

**get\_public\_key()**

Returns the pubkey

**point()**

Return the point for the public key

**unCompressed()**

Derive uncompressed key

## beemgraphenebase.base58

**class** beemgraphenebase.base58.**Base58** (*data*, *prefix*='GPH')

Bases: object

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.

beemgraphenebase.base58.**b58decode** (*v*)

`beemgraphenebase.base58.b58encode (v)`  
`beemgraphenebase.base58.base58CheckDecode (s)`  
`beemgraphenebase.base58.base58CheckEncode (version, payload)`  
`beemgraphenebase.base58.base58decode (base58_str)`  
`beemgraphenebase.base58.base58encode (hexstring)`  
`beemgraphenebase.base58.doublesha256 (s)`  
`beemgraphenebase.base58.gphBase58CheckDecode (s)`  
`beemgraphenebase.base58.gphBase58CheckEncode (s)`  
`beemgraphenebase.base58.log = <logging.Logger object>`  
Default Prefix  
`beemgraphenebase.base58.ripemd160 (s)`

### beemgraphenebase.bip38

**exception** `beemgraphenebase.bip38.SaltException`  
Bases: Exception

`beemgraphenebase.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)

`beemgraphenebase.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`

### beemgraphenebase.ecdsasig

`beemgraphenebase.ecdsasig.compressedPubkey (pk)`

`beemgraphenebase.ecdsasig.recoverPubkeyParameter (message, digest, signature, pubkey)`  
Use to derive a number that allows to easily recover the public key from the signature

`beemgraphenebase.ecdsasig.recover_public_key (digest, signature, i, message=None)`  
Recover the public key from the the signature



`beemgraphenebase.ecdsasig.sign_message` (*message*, *wif*, *hashfn=<built-in function openssl\_sha256>*)

Sign a digest with a wif key

**Parameters** *wif* (*str*) – Private key in

`beemgraphenebase.ecdsasig.verify_message` (*message*, *signature*, *hashfn=<built-in function openssl\_sha256>*, *recover\_parameter=None*)

## beemgraphenebase.objects

**class** `beemgraphenebase.objects.GrapheneObject` (*data=None*)

Bases: `object`

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

`json()`

`toJson()`

**class** `beemgraphenebase.objects.Operation` (*op*)

Bases: `object`

`getOperationNameForId` (*i*)

Convert an operation id into the corresponding string

`operations()`

`beemgraphenebase.objects.isArgsThisClass` (*self*, *args*)

## beemgraphenebase.objecttypes

`beemgraphenebase.objecttypes.object_type` = {'OBJECT\_TYPE\_COUNT': 3, 'account': 2, 'base': 1}

Object types for object ids

## beemgraphenebase.operations

`beemgraphenebase.operationids.operations` = {'demooperation': 0}

Operation ids

## beemgraphenebase.signedtransactions

**class** `beemgraphenebase.signedtransactions.Signed_Transaction` (*\*args*, *\*\*kwargs*)

Bases: `beemgraphenebase.objects.GrapheneObject`

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

**deriveDigest** (*chain*)

**getChainParams** (*chain*)

**getKnownChains** ()

**getOperationKlass** ()

**id**

The transaction id of this transaction

**sign** (*wifkeys*, *chain=None*)

Sign the transaction with the provided private keys.

#### Parameters

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

**verify** (*pubkeys=[]*, *chain=None*, *recover\_parameter=False*)

Returned pubkeys have to be checked if they are existing

## 3.8 Contributing to beem

We welcome your contributions to our project.

### 3.8.1 Repository

The repository of beem is currently located at:

<https://github.com/holgern/beem>

### 3.8.2 Flow

This project makes heavy use of [git flow](#). If you are not familiar with it, then the most important thing for your to understand is that:

pull requests need to be made against the develop branch

### 3.8.3 How to Contribute

0. Familiarize yourself with *contributing on github* <<https://guides.github.com/activities/contributing-to-open-source/>>
1. Fork or branch from the master.
2. Create commits following the commit style
3. Start a pull request to the master branch
4. Wait for a @holger80 or another member to review

### 3.8.4 Issues

Feel free to submit issues and enhancement requests.

### 3.8.5 Contributing

Please refer to each project's style guidelines and guidelines for submitting patches and additions. In general, we follow the “fork-and-pull” Git workflow.

1. **Fork** the repo on GitHub
2. **Clone** the project to your own machine
3. **Commit** changes to your own branch
4. **Push** your work back up to your fork
5. Submit a **Pull request** so that we can review your changes

NOTE: Be sure to merge the latest from “upstream” before making a pull request!

### 3.8.6 Copyright and Licensing

This library is open sources under the MIT license. We require your to release your code under that license as well.

## 3.9 Support and Questions

Help and discussion channel for beem can be found here:

- <https://discord.gg/4HM592V>

## 3.10 Indices and Tables

- [genindex](#)
- [modindex](#)



## CHAPTER 4

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`



### b

- `beem.account`, 63
- `beem.aes`, 82
- `beem.amount`, 83
- `beem.asciichart`, 82
- `beem.asset`, 85
- `beem.block`, 99
- `beem.blockchain`, 101
- `beem.blockchainobject`, 106
- `beem.comment`, 107
- `beem.conveyor`, 112
- `beem.discussions`, 114
- `beem.exceptions`, 120
- `beem.imageuploader`, 122
- `beem.instance`, 122
- `beem.market`, 123
- `beem.memo`, 128
- `beem.message`, 131
- `beem.nodelist`, 97
- `beem.notify`, 131
- `beem.price`, 132
- `beem.rc`, 134
- `beem.snapshot`, 135
- `beem.steem`, 85
- `beem.steemconnect`, 98
- `beem.storage`, 137
- `beem.transactionbuilder`, 140
- `beem.utils`, 142
- `beem.vote`, 143
- `beem.wallet`, 145
- `beem.witness`, 148
- `beemapi.exceptions`, 151
- `beemapi.graphenerpc`, 155
- `beemapi.node`, 154
- `beemapi.steemnoderpc`, 150
- `beembase.memo`, 157
- `beembase.objects`, 158
- `beembase.objecttypes`, 159
- `beembase.operationids`, 159
- `beembase.signedtransactions`, 160
- `beembase.transactions`, 160
- `beemgraphenebase.account`, 160
- `beemgraphenebase.base58`, 163
- `beemgraphenebase.bip38`, 164
- `beemgraphenebase.ecdsasig`, 164
- `beemgraphenebase.objects`, 165
- `beemgraphenebase.objecttypes`, 165
- `beemgraphenebase.operationids`, 165
- `beemgraphenebase.signedtransactions`, 165





## Symbols

- account\_creation\_fee <account\_creation\_fee>
  - beempy-witnesscreate command line option, 44
  - beempy-witnessproperties command line option, 46
  - beempy-witnessupdate command line option, 46
- account\_subsidy\_budget <account\_subsidy\_budget>
  - beempy-witnessproperties command line option, 46
- account\_subsidy\_decay <account\_subsidy\_decay>
  - beempy-witnessproperties command line option, 46
- ascii
  - beempy-orderbook command line option, 33
  - beempy-pricehistory command line option, 36
  - beempy-tradehistory command line option, 39
- auto\_vest
  - beempy-powerdownroute command line option, 35
- chart
  - beempy-orderbook command line option, 32
- claim\_all\_sbd
  - beempy-claimreward command line option, 22
- claim\_all\_steem
  - beempy-claimreward command line option, 22
- claim\_all\_vests
  - beempy-claimreward command line option, 22
- confirm
  - beempy-delkey command line option, 25
  - beempy-deltoken command line option, 26
- direction <direction>
  - beempy-votes command line option, 43
- fee <fee>
  - beempy-newaccount command line option, 31
- file <file>
  - beempy-broadcast command line option, 21
  - beempy-sign command line option, 39
- hours <hours>
  - beempy-tradehistory command line option, 39
- import-brain-key
  - beempy-keygen command line option, 30
- key <key>
  - beempy-updatememokey command line option, 41
- limit <limit>
  - beempy-witnesses command line option, 45
- maximum\_block\_size <maximum\_block\_size>
  - beempy-witnesscreate command line option, 44
  - beempy-witnessproperties command line option, 46
  - beempy-witnessupdate command line option, 46
- new\_signing\_key <new\_signing\_key>
  - beempy-witnessproperties command line option, 46
- orderid <orderid>
  - beempy-buy command line option, 21
  - beempy-sell command line option, 37
- payout <payout>
  - beempy-curation command line option, 24
- percentage <percentage>
  - beempy-powerdownroute command line option, 35
- permission <permission>
  - beempy-allow command line option, 20
  - beempy-disallow command line option, 26
- raw
  - beempy-pingnode command line option, 34
- remove
  - beempy-pingnode command line option, 34
- results
  - beempy-nextnode command line option, 32
- reward\_sbd <reward\_sbd>
  - beempy-claimreward command line option, 22
- reward\_steem <reward\_steem>
  - beempy-claimreward command line option, 22
- reward\_vests <reward\_vests>
  - beempy-claimreward command line option, 22
- roles <roles>
  - beempy-importaccount command line option, 29
- sbd\_interest\_rate <sbd\_interest\_rate>
  - beempy-witnesscreate command line option, 44
  - beempy-witnessproperties command line option, 46
  - beempy-witnessupdate command line option, 46
- sequence <sequence>
  - beempy-keygen command line option, 30
- show-date
  - beempy-orderbook command line option, 32

- `--signing_key <signing_key>`
  - beem-py-witnessupdate command line option, 47
- `--sort`
  - beem-py-pingnode command line option, 34
- `--support-peg`
  - beem-py-witnessfeed command line option, 45
- `--test-unlock`
  - beem-py-walletinfo command line option, 43
- `--threading`
  - beem-py-pingnode command line option, 34
- `--threshold <threshold>`
  - beem-py-allow command line option, 20
  - beem-py-disallow command line option, 26
- `--to <to>`
  - beem-py-powerup command line option, 36
- `--unsafe-import-key <unsafe_import_key>`
  - beem-py-addkey command line option, 19
  - beem-py-parsewif command line option, 33
- `--unsafe-import-token <unsafe_import_token>`
  - beem-py-addtoken command line option, 19
- `--url`
  - beem-py-currentnode command line option, 25
- `--url <url>`
  - beem-py-witnesscreate command line option, 44
  - beem-py-witnessproperties command line option, 46
  - beem-py-witnessupdate command line option, 47
- `--version`
  - beem-py command line option, 19
  - beem-py-currentnode command line option, 25
- `--weight <weight>`
  - beem-py-allow command line option, 20
- `--what <what>`
  - beem-py-follow command line option, 28
  - beem-py-mute command line option, 31
- `--wipe`
  - beem-py-createwallet command line option, 23
- `--witness <witness>`
  - beem-py-witnessupdate command line option, 46
- `-a, --account <account>`
  - beem-py-allow command line option, 20
  - beem-py-approvewitness command line option, 20
  - beem-py-buy command line option, 21
  - beem-py-cancel command line option, 22
  - beem-py-convert command line option, 23
  - beem-py-curation command line option, 24
  - beem-py-delegate command line option, 25
  - beem-py-delpoint command line option, 26
  - beem-py-disallow command line option, 26
  - beem-py-disapprovewitness command line option, 27
  - beem-py-downvote command line option, 27
  - beem-py-follow command line option, 28
  - beem-py-mute command line option, 31
  - beem-py-newaccount command line option, 31
  - beem-py-powerdown command line option, 35
  - beem-py-powerdownroute command line option, 35
  - beem-py-powerup command line option, 36
  - beem-py-resteem command line option, 36
  - beem-py-sell command line option, 37
  - beem-py-setprofile command line option, 38
  - beem-py-transfer command line option, 40
  - beem-py-unfollow command line option, 40
  - beem-py-updatememokey command line option, 41
  - beem-py-upvote command line option, 41
- `-a, --author`
  - beem-py-pending command line option, 33
  - beem-py-rewards command line option, 37
- `-a, --only-appbase`
  - beem-py-updatenodes command line option, 41
- `-b, --base <base>`
  - beem-py-witnessfeed command line option, 45
- `-c, --comment`
  - beem-py-pending command line option, 33
  - beem-py-rewards command line option, 37
- `-d, --days <days>`
  - beem-py-curation command line option, 24
  - beem-py-pending command line option, 33
  - beem-py-rewards command line option, 37
  - beem-py-tradehistory command line option, 39
  - beem-py-votes command line option, 43
- `-d, --no-broadcast`
  - beem-py command line option, 18
- `-e, --expires <expires>`
  - beem-py command line option, 19
- `-e, --export <export>`
  - beem-py-curation command line option, 24
  - beem-py-votes command line option, 43
- `-e, --permlink`
  - beem-py-pending command line option, 33
  - beem-py-rewards command line option, 37
- `-h, --height <height>`
  - beem-py-orderbook command line option, 33
  - beem-py-pricehistory command line option, 36
  - beem-py-tradehistory command line option, 39
- `-h, --only-https`
  - beem-py-updatenodes command line option, 41
- `-i, --incoming`
  - beem-py-votes command line option, 43
- `-i, --sbd-to-steem`
  - beem-py-ticker command line option, 39
  - beem-py-tradehistory command line option, 39
- `-l, --create-link`
  - beem-py command line option, 19
- `-l, --length <length>`
  - beem-py-curation command line option, 24
  - beem-py-pending command line option, 33
  - beem-py-rewards command line option, 37
- `-l, --limit <limit>`
  - beem-py-orderbook command line option, 32

beem-pending command line option, 33  
 beem-rewards command line option, 37  
 beem-witnessfeed command line option, 45  
 beem-pending command line option, 33  
 beem-rewards command line option, 37  
 beem-curation command line option, 24  
 beem-updatenodes command line option, 41  
 beem-signing-account <signing\_account>  
 beem-featureflags command line option, 28  
 beem-userdata command line option, 42  
 beem-command line option, 19  
 beem-updatenodes command line option, 41  
 beem-curation command line option, 24  
 beem-pending command line option, 33  
 beem-rewards command line option, 37  
 beem-verify command line option, 42  
 beem-verify command line option, 42  
 beem-curation  
 beem-pending command line option, 33  
 beem-rewards command line option, 37  
 beem-min-vote <min\_vote>  
 beem-curation command line option, 24  
 beem-verbose <verbose>  
 beem-command line option, 19  
 beem-max-vote <max\_vote>  
 beem-curation command line option, 24  
 beem-only-wss

beem-updatenodes command line option, 41  
 beem-weight <weight>  
 beem-downvote command line option, 27  
 beem-upvote command line option, 41  
 beem-width <width>  
 beem-orderbook command line option, 32  
 beem-pricehistory command line option, 36  
 beem-tradehistory command line option, 39  
 beem-min-performance <min\_performance>  
 beem-curation command line option, 24  
 beem-unsigned  
 beem-command line option, 19  
 beem-max-performance <max\_performance>  
 beem-curation command line option, 24  
 SteemWebsocket\_\_set\_subscriptions()  
 (beemapi.websocket.SteamWebsocket  
 method), 153  
 \_\_events\_\_ (beemapi.websocket.SteamWebsocket  
 attribute), 153  
 \_\_getattr\_\_() (beemapi.websocket.SteamWebsocket  
 method), 153  
 \_\_init\_\_() (beemapi.websocket.SteamWebsocket  
 method), 153  
 \_\_module\_\_ (beemapi.websocket.SteamWebsocket  
 attribute), 153  
 \_\_ping\_\_() (beemapi.websocket.SteamWebsocket  
 method), 153

## A

abort() (beem.blockchain.Pool method), 106

### ACCOUNT

beem-balance command line option, 21  
 beem-claimreward command line option, 23  
 beem-featureflags command line option, 28  
 beem-follower command line option, 28  
 beem-following command line option, 29  
 beem-importaccount command line option, 29  
 beem-interest command line option, 30  
 beem-muter command line option, 31  
 beem-muting command line option, 31  
 beem-openorders command line option, 32  
 beem-permissions command line option, 34  
 beem-power command line option, 34  
 beem-userdata command line option, 42  
 beem-votes command line option, 43  
 beem-witnesses command line option, 45

account (beem.witness.Witness attribute), 149

Account (class in beem.account), 63

account\_update\_dict() (beem.rc.RC method), 134

AccountDoesNotExistException, 120

AccountExistsException, 120

### ACCOUNTNAME

beem-newaccount command line option, 32

accountopenorders() (beem.market.Market method), 123

## ACCOUNTS

- beem-pending command line option, [34](#)
- beem-rewards command line option, [37](#)
- Accounts (class in `beem.account`), [81](#)
- AccountSnapshot (class in `beem.snapshot`), [135](#)
- AccountsObject (class in `beem.account`), [81](#)
- AccountVotes (class in `beem.vote`), [143](#)
- ActiveVotes (class in `beem.vote`), [144](#)
- `adapt_on_series()` (`beem.asciichart.AsciiChart` method), [82](#)
- `add()` (`beem.storage.Key` method), [138](#)
- `add()` (`beem.storage.Token` method), [139](#)
- `add_axis()` (`beem.asciichart.AsciiChart` method), [82](#)
- `add_curve()` (`beem.asciichart.AsciiChart` method), [82](#)
- `add_custom_chains()` (`beem-base.signedtransactions.Signed_Transaction` method), [160](#)
- `addPrivateKey()` (`beem.wallet.Wallet` method), [145](#)
- Address (class in `beemgraphenebase.account`), [160](#)
- `addSigningInformation()` (`beem.transactionbuilder.TransactionBuilder` method), [140](#)
- `addToken()` (`beem.wallet.Wallet` method), [145](#)
- `addTzInfo()` (in module `beem.utils`), [142](#)
- AESCipher (class in `beem.aes`), [82](#)
- `alive()` (`beem.blockchain.Pool` method), [106](#)
- `allow()` (`beem.account.Account` method), [63](#)

AMOUNT

- beem-buy command line option, [21](#)
- beem-convert command line option, [23](#)
- beem-delegate command line option, [25](#)
- beem-powerdown command line option, [35](#)
- beem-powerup command line option, [36](#)
- beem-sell command line option, [38](#)
- beem-transfer command line option, [40](#)
- amount (`beem.amount.Amount` attribute), [84](#)
- Amount (class in `beem.amount`), [83](#)
- Amount (class in `beembase.objects`), [158](#)
- ApiNotSupported, [151](#)
- `appauthor` (`beem.storage.DataDir` attribute), [137](#)
- `appendMissingSignatures()` (`beem.transactionbuilder.TransactionBuilder` method), [141](#)
- `appendOps()` (`beem.transactionbuilder.TransactionBuilder` method), [141](#)
- `appendSigner()` (`beem.transactionbuilder.TransactionBuilder` method), [141](#)
- `appendWif()` (`beem.transactionbuilder.TransactionBuilder` method), [141](#)
- `appname` (`beem.storage.DataDir` attribute), [137](#)
- `approvewitness()` (`beem.account.Account` method), [64](#)
- `as_base()` (`beem.price.Price` method), [133](#)
- `as_quote()` (`beem.price.Price` method), [133](#)
- AsciiChart (class in `beem.asciichart`), [82](#)

## ASSET

- beem-buy command line option, [21](#)
- beem-sell command line option, [38](#)
- beem-transfer command line option, [40](#)
- asset (`beem.amount.Amount` attribute), [84](#)
- asset (`beem.asset.Asset` attribute), [85](#)
- Asset (class in `beem.asset`), [85](#)
- AssetDoesNotExistException, [120](#)
- `assets_from_string()` (in module `beem.utils`), [142](#)
- author (`beem.comment.Comment` attribute), [107](#)

AUTHORPERM

- beem-curation command line option, [24](#)
- authorperm (`beem.comment.Comment` attribute), [107](#)
- authorperm (`beem.vote.Vote` attribute), [144](#)
- available\_balances (`beem.account.Account` attribute), [64](#)
- `awaitTxConfirmation()` (`beem.blockchain.Blockchain` method), [102](#)

## B

- `b58decode()` (in module `beemgraphenebase.base58`), [163](#)
- `b58encode()` (in module `beemgraphenebase.base58`), [163](#)
- balances (`beem.account.Account` attribute), [64](#)
- Base58 (class in `beemgraphenebase.base58`), [163](#)
- `base58CheckDecode()` (in module `beem-graphenebase.base58`), [164](#)
- `base58CheckEncode()` (in module `beem-graphenebase.base58`), [164](#)
- `base58decode()` (in module `beemgraphenebase.base58`), [164](#)
- `base58encode()` (in module `beemgraphenebase.base58`), [164](#)
- BatchedCallsNotSupported, [120](#)
- `beem.account` (module), [63](#)
- `beem.aes` (module), [82](#)
- `beem.amount` (module), [83](#)
- `beem.asciichart` (module), [82](#)
- `beem.asset` (module), [85](#)
- `beem.block` (module), [99](#)
- `beem.blockchain` (module), [101](#)
- `beem.blockchainobject` (module), [106](#)
- `beem.comment` (module), [107](#)
- `beem.conveyor` (module), [112](#)
- `beem.discussions` (module), [114](#)
- `beem.exceptions` (module), [120](#)
- `beem.imageuploader` (module), [122](#)
- `beem.instance` (module), [122](#)
- `beem.market` (module), [123](#)
- `beem.memo` (module), [128](#)
- `beem.message` (module), [131](#)
- `beem.nodelist` (module), [97](#)
- `beem.notify` (module), [131](#)
- `beem.price` (module), [132](#)
- `beem.rc` (module), [134](#)
- `beem.snapshot` (module), [135](#)

- beem.steem (module), 85
- beem.steemconnect (module), 98
- beem.storage (module), 137
- beem.transactionbuilder (module), 140
- beem.utils (module), 142
- beem.vote (module), 143
- beem.wallet (module), 145
- beem.witness (module), 148
- beemapi.exceptions (module), 151
- beemapi.graphenerpc (module), 155
- beemapi.node (module), 154
- beemapi.steemnodeRPC (module), 150
- beembase.memo (module), 157
- beembase.objects (module), 158
- beembase.objecttypes (module), 159
- beembase.operationids (module), 159
- beembase.signedtransactions (module), 160
- beembase.transactions (module), 160
- beemgraphenebase.account (module), 160
- beemgraphenebase.base58 (module), 163
- beemgraphenebase.bip38 (module), 164
- beemgraphenebase.ecdsasig (module), 164
- beemgraphenebase.objects (module), 165
- beemgraphenebase.objecttypes (module), 165
- beemgraphenebase.operationids (module), 165
- beemgraphenebase.signedtransactions (module), 165
- beempy command line option
  - version, 19
  - d, -no-broadcast, 18
  - e, -expires <expires>, 19
  - l, -create-link, 19
  - n, -node <node>, 18
  - o, -offline, 18
  - p, -no-wallet, 18
  - s, -steemconnect, 19
  - v, -verbose <verbose>, 19
  - x, -unsigned, 19
- beempy-addkey command line option
  - unsafe-import-key <unsafe\_import\_key>, 19
- beempy-addtoken command line option
  - unsafe-import-token <unsafe\_import\_token>, 19
  - NAME, 19
- beempy-allow command line option
  - permission <permission>, 20
  - threshold <threshold>, 20
  - weight <weight>, 20
  - a, -account <account>, 20
  - FOREIGN\_ACCOUNT, 20
- beempy-approvewitness command line option
  - a, -account <account>, 20
  - WITNESS, 20
- beempy-balance command line option
  - ACCOUNT, 21
- beempy-broadcast command line option
  - file <file>, 21
- beempy-buy command line option
  - orderid <orderid>, 21
  - a, -account <account>, 21
  - AMOUNT, 21
  - ASSET, 21
  - PRICE, 21
- beempy-cancel command line option
  - a, -account <account>, 22
  - ORDERID, 22
- beempy-claimreward command line option
  - claim\_all\_sbd, 22
  - claim\_all\_steem, 22
  - claim\_all\_vests, 22
  - reward\_sbd <reward\_sbd>, 22
  - reward\_steem <reward\_steem>, 22
  - reward\_vests <reward\_vests>, 22
  - ACCOUNT, 23
- beempy-convert command line option
  - a, -account <account>, 23
  - AMOUNT, 23
- beempy-createwallet command line option
  - wipe, 23
- beempy-curation command line option
  - payout <payout>, 24
  - a, -account <account>, 24
  - d, -days <days>, 24
  - e, -export <export>, 24
  - l, -length <length>, 24
  - m, -limit <limit>, 24
  - p, -permalink, 24
  - s, -short, 24
  - t, -title, 24
  - v, -min-vote <min\_vote>, 24
  - w, -max-vote <max\_vote>, 24
  - x, -min-performance <min\_performance>, 24
  - y, -max-performance <max\_performance>, 24
  - AUTHORPERM, 24
- beempy-currentnode command line option
  - url, 25
  - version, 25
- beempy-delegate command line option
  - a, -account <account>, 25
  - AMOUNT, 25
  - TO\_ACCOUNT, 25
- beempy-delkey command line option
  - confirm, 25
  - PUB, 25
- beempy-delfprofile command line option
  - a, -account <account>, 26
  - VARIABLE, 26
- beempy-deltoken command line option
  - confirm, 26
  - NAME, 26

beempy-disallow command line option

-permission <permission>, 26  
-threshold <threshold>, 26  
-a, -account <account>, 26  
FOREIGN\_ACCOUNT, 27

beempy-disapprovewitness command line option

-a, -account <account>, 27  
WITNESS, 27

beempy-downvote command line option

-a, -account <account>, 27  
-w, -weight <weight>, 27  
POST, 27  
VOTE\_WEIGHT, 27

beempy-featureflags command line option

-s, -signing-account <signing\_account>, 28  
ACCOUNT, 28

beempy-follow command line option

-what <what>, 28  
-a, -account <account>, 28  
FOLLOW, 28

beempy-follower command line option

ACCOUNT, 28

beempy-following command line option

ACCOUNT, 29

beempy-importaccount command line option

-roles <roles>, 29  
ACCOUNT, 29

beempy-info command line option

OBJECTS, 29

beempy-interest command line option

ACCOUNT, 30

beempy-keygen command line option

-import-brain-key, 30  
-sequence <sequence>, 30

beempy-mute command line option

-what <what>, 31  
-a, -account <account>, 31  
MUTE, 31

beempy-muter command line option

ACCOUNT, 31

beempy-muting command line option

ACCOUNT, 31

beempy-newaccount command line option

-fee <fee>, 31  
-a, -account <account>, 31  
ACCOUNTNAME, 32

beempy-nextnode command line option

-results, 32

beempy-openorders command line option

ACCOUNT, 32

beempy-orderbook command line option

-ascii, 33  
-chart, 32  
-show-date, 32

-h, -height <height>, 33

-l, -limit <limit>, 32

-w, -width <width>, 32

beempy-parsewif command line option

-unsafe-import-key <unsafe\_import\_key>, 33

beempy-pending command line option

-a, -author, 33  
-c, -comment, 33  
-d, -days <days>, 33  
-e, -permlink, 33  
-l, -length <length>, 33  
-p, -post, 33  
-s, -only-sum, 33  
-t, -title, 33  
-v, -curation, 33  
ACCOUNTS, 34

beempy-permissions command line option

ACCOUNT, 34

beempy-pingnode command line option

-raw, 34  
-remove, 34  
-sort, 34  
-threading, 34

beempy-power command line option

ACCOUNT, 34

beempy-powerdown command line option

-a, -account <account>, 35  
AMOUNT, 35

beempy-powerdownroute command line option

-auto\_vest, 35  
-percentage <percentage>, 35  
-a, -account <account>, 35  
TO, 35

beempy-powerup command line option

-to <to>, 36  
-a, -account <account>, 36  
AMOUNT, 36

beempy-pricehistory command line option

-ascii, 36  
-h, -height <height>, 36  
-w, -width <width>, 36

beempy-resteem command line option

-a, -account <account>, 36  
IDENTIFIER, 36

beempy-rewards command line option

-a, -author, 37  
-c, -comment, 37  
-d, -days <days>, 37  
-e, -permlink, 37  
-l, -length <length>, 37  
-p, -post, 37  
-s, -only-sum, 37  
-t, -title, 37  
-v, -curation, 37



- ACCOUNTS, 37
- beempy-sell command line option
  - orderid <orderid>, 37
  - a, -account <account>, 37
  - AMOUNT, 38
  - ASSET, 38
  - PRICE, 38
- beempy-set command line option
  - KEY, 38
  - VALUE, 38
- beempy-setprofile command line option
  - a, -account <account>, 38
  - p, -pair <pair>, 38
  - VALUE, 38
  - VARIABLE, 38
- beempy-sign command line option
  - file <file>, 39
- beempy-ticker command line option
  - i, -sbd-to-steem, 39
- beempy-tradehistory command line option
  - ascii, 39
  - hours <hours>, 39
  - d, -days <days>, 39
  - h, -height <height>, 39
  - i, -sbd-to-steem, 39
  - l, -limit <limit>, 39
  - w, -width <width>, 39
- beempy-transfer command line option
  - a, -account <account>, 40
  - AMOUNT, 40
  - ASSET, 40
  - MEMO, 40
  - TO, 40
- beempy-unfollow command line option
  - a, -account <account>, 40
  - UNFOLLOW, 40
- beempy-updatememokey command line option
  - key <key>, 41
  - a, -account <account>, 41
- beempy-updatenodes command line option
  - a, -only-appbase, 41
  - h, -only-https, 41
  - n, -only-non-appbase, 41
  - s, -show, 41
  - t, -test, 41
  - w, -only-wss, 41
- beempy-upvote command line option
  - a, -account <account>, 41
  - w, -weight <weight>, 41
  - POST, 42
  - VOTE\_WEIGHT, 42
- beempy-userdata command line option
  - s, -signing-account <signing\_account>, 42
  - ACCOUNT, 42
- beempy-verify command line option
  - t, -trx <trx>, 42
  - u, -use-api, 42
  - BLOCKNUMBER, 42
- beempy-votes command line option
  - direction <direction>, 43
  - d, -days <days>, 43
  - e, -export <export>, 43
  - i, -incoming, 43
  - o, -outgoing, 43
  - ACCOUNT, 43
- beempy-walletinfo command line option
  - test-unlock, 43
- beempy-witness command line option
  - WITNESS, 43
- beempy-witnesscreate command line option
  - account\_creation\_fee <account\_creation\_fee>, 44
  - maximum\_block\_size <maximum\_block\_size>, 44
  - sbd\_interest\_rate <sbd\_interest\_rate>, 44
  - url <url>, 44
  - PUB\_SIGNING\_KEY, 44
  - WITNESS, 44
- beempy-witnessdisable command line option
  - WITNESS, 44
- beempy-witnessenable command line option
  - SIGNING\_KEY, 44
  - WITNESS, 44
- beempy-witnesses command line option
  - limit <limit>, 45
  - ACCOUNT, 45
- beempy-witnessfeed command line option
  - support-peg, 45
  - b, -base <base>, 45
  - q, -quote <quote>, 45
  - WIF, 45
  - WITNESS, 45
- beempy-witnessproperties command line option
  - account\_creation\_fee <account\_creation\_fee>, 46
  - account\_subsidy\_budget <account\_subsidy\_budget>, 46
  - account\_subsidy\_decay <account\_subsidy\_decay>, 46
  - maximum\_block\_size <maximum\_block\_size>, 46
  - new\_signing\_key <new\_signing\_key>, 46
  - sbd\_interest\_rate <sbd\_interest\_rate>, 46
  - url <url>, 46
  - WIF, 46
  - WITNESS, 46
- beempy-witnessupdate command line option
  - account\_creation\_fee <account\_creation\_fee>, 46
  - maximum\_block\_size <maximum\_block\_size>, 46
  - sbd\_interest\_rate <sbd\_interest\_rate>, 46
  - signing\_key <signing\_key>, 47
  - url <url>, 47

- witness <witness>, [46](#)
- Beneficiaries (class in beembase.objects), [158](#)
- Beneficiary (class in beembase.objects), [158](#)
- Block (class in beem.block), [99](#)
- block\_num (beem.block.Block attribute), [100](#)
- block\_num (beem.block.BlockHeader attribute), [101](#)
- block\_time() (beem.blockchain.Blockchain method), [102](#)
- block\_timestamp() (beem.blockchain.Blockchain method), [102](#)
- Blockchain (class in beem.blockchain), [101](#)
- BlockchainObject (class in beem.blockchainobject), [106](#)
- BlockDoesNotExistException, [120](#)
- BlockHeader (class in beem.block), [100](#)
- BLOCKNUMBER
  - beem-verify command line option, [42](#)
- blocks() (beem.blockchain.Blockchain method), [102](#)
- BlockWaitTimeExceeded, [120](#)
- blog\_history() (beem.account.Account method), [64](#)
- body (beem.comment.Comment attribute), [107](#)
- BrainKey (class in beemgraphenebase.account), [161](#)
- broadcast() (beem.steem.Steem method), [87](#)
- broadcast() (beem.steemconnect.SteemConnect method), [98](#)
- broadcast() (beem.transactionbuilder.TransactionBuilder method), [141](#)
- btc\_usd\_ticker() (beem.market.Market static method), [124](#)
- build() (beem.snapshot.AccountSnapshot method), [136](#)
- build\_curation\_arrays() (beem.snapshot.AccountSnapshot method), [136](#)
- build\_rep\_arrays() (beem.snapshot.AccountSnapshot method), [136](#)
- build\_sp\_arrays() (beem.snapshot.AccountSnapshot method), [136](#)
- build\_vp\_arrays() (beem.snapshot.AccountSnapshot method), [136](#)
- buy() (beem.market.Market method), [124](#)

## C

- cache() (beem.blockchainobject.BlockchainObject method), [106](#)
- CallRetriesReached, [151](#)
- cancel() (beem.market.Market method), [124](#)
- cancel\_subscriptions() (beemapi.websocket.SteemWebsocket method), [153](#)
- cancel\_transfer\_from\_savings() (beem.account.Account method), [64](#)
- category (beem.comment.Comment attribute), [107](#)
- chain\_params (beem.steem.Steem attribute), [87](#)
- changePassphrase() (beem.wallet.Wallet method), [145](#)
- changePassword() (beem.storage.MasterPassword method), [139](#)
- check\_asset() (in module beem.amount), [84](#)

- checkBackup() (beem.storage.Configuration method), [137](#)
- child() (beemgraphenebase.account.PrivateKey method), [162](#)
- claim\_account() (beem.rc.RC method), [134](#)
- claim\_account() (beem.steem.Steem method), [87](#)
- claim\_reward\_balance() (beem.account.Account method), [65](#)
- clean\_data() (beem.storage.DataDir method), [138](#)
- clear() (beem.steem.Steem method), [87](#)
- clear() (beem.transactionbuilder.TransactionBuilder method), [141](#)
- clear\_cache() (beem.blockchainobject.BlockchainObject static method), [106](#)
- clear\_cache() (in module beem.instance), [122](#)
- clear\_cache\_from\_expired\_items() (beem.blockchainobject.BlockchainObject method), [107](#)
- clear\_data() (beem.asciichart.AsciiChart method), [83](#)
- clear\_expired\_items() (beem.blockchainobject.ObjectCache method), [107](#)
- clear\_local\_keys() (beem.wallet.Wallet method), [146](#)
- clear\_local\_token() (beem.wallet.Wallet method), [146](#)
- clearWifs() (beem.transactionbuilder.TransactionBuilder method), [141](#)
- close() (beem.notify.Notify method), [131](#)
- close() (beemapi.websocket.SteemWebsocket method), [153](#)
- Comment (class in beem.comment), [107](#)
- comment() (beem.rc.RC method), [134](#)
- comment\_dict() (beem.rc.RC method), [134](#)
- Comment\_discussions\_by\_payout (class in beem.discussions), [114](#)
- comment\_history() (beem.account.Account method), [65](#)
- comment\_options() (beem.steem.Steem method), [87](#)
- CommentOptionExtensions (class in beembase.objects), [158](#)
- compressed() (beemgraphenebase.account.PublicKey method), [163](#)
- compressedpubkey() (beemgraphenebase.account.PrivateKey method), [162](#)
- compressedPubkey() (in module beemgraphenebase.ecdsasig), [164](#)
- config (beem.instance.SharedInstance attribute), [122](#)
- config\_defaults (beem.storage.Configuration attribute), [137](#)
- config\_key (beem.storage.MasterPassword attribute), [139](#)
- configStorage (beem.wallet.Wallet attribute), [146](#)
- Configuration (class in beem.storage), [137](#)
- connect() (beem.steem.Steem method), [87](#)
- construct\_authorperm() (in module beem.utils), [142](#)
- construct\_authorpermvoter() (in module beem.utils), [142](#)
- constructTx() (beem.transactionbuilder.TransactionBuilder



- method), 141
- ContentDoesNotExistException, 120
- convert() (beem.account.Account method), 65
- Conveyor (class in beem.conveyor), 112
- copy() (beem.amount.Amount method), 84
- copy() (beem.price.Price method), 133
- create() (beem.wallet.Wallet method), 146
- create\_account() (beem.steem.Steem method), 87
- create\_claimed\_account() (beem.steem.Steem method), 89
- create\_hot\_sign\_url() (beem.steemconnect.SteemConnect method), 99
- create\_table() (beem.storage.Configuration method), 137
- create\_table() (beem.storage.Key method), 138
- create\_table() (beem.storage.Token method), 139
- create\_ws\_instance() (in module beemapi.graphenepc), 156
- created() (beem.wallet.Wallet method), 146
- curation\_penalty\_compensation\_SBD() (beem.comment.Comment method), 107
- curation\_stats() (beem.account.Account method), 65
- custom\_json() (beem.rc.RC method), 134
- custom\_json() (beem.steem.Steem method), 90
- custom\_json\_dict() (beem.rc.RC method), 134
- ## D
- data\_dir (beem.storage.DataDir attribute), 138
- DataDir (class in beem.storage), 137
- decode\_memo() (in module beembase.memo), 157
- decode\_memo\_bts() (in module beembase.memo), 157
- decodeRPCErrorMsg() (in module beemapi.exceptions), 152
- decrypt() (beem.aes.AESCipher method), 82
- decrypt() (beem.memo.Memo method), 130
- decrypt() (in module beemgraphenebase.bip38), 164
- decrypt\_token() (beem.wallet.Wallet method), 146
- decrypt\_wif() (beem.wallet.Wallet method), 146
- decrypted\_master (beem.storage.MasterPassword attribute), 139
- decryptEncryptedMaster() (beem.storage.MasterPassword method), 139
- default\_handler() (in module beem.blockchain), 106
- delegate\_vesting\_shares() (beem.account.Account method), 66
- delete() (beem.comment.Comment method), 108
- delete() (beem.storage.Configuration method), 137
- delete() (beem.storage.Key method), 138
- delete() (beem.storage.Token method), 139
- depth (beem.comment.Comment attribute), 108
- derive256address\_with\_version() (beem-graphenebase.account.Address method), 161
- derive\_from\_seed() (beem-graphenebase.account.PrivateKey method), 162
- derive\_permalink() (in module beem.utils), 142
- derive\_private\_key() (beem-graphenebase.account.PrivateKey method), 162
- deriveChecksum() (beem.storage.MasterPassword method), 139
- deriveChecksum() (beem.wallet.Wallet method), 146
- deriveDigest() (beemgraphenebase.signedtransactions.Signed\_Transaction method), 166
- derivesha256address() (beem-graphenebase.account.Address method), 161
- derivesha512address() (beem-graphenebase.account.Address method), 161
- derSigToHexSig() (beem-graphenebase.signedtransactions.Signed\_Transaction method), 166
- disallow() (beem.account.Account method), 66
- disapprovewitness() (beem.account.Account method), 66
- Discussions (class in beem.discussions), 114
- Discussions\_by\_active (class in beem.discussions), 115
- Discussions\_by\_author\_before\_date (class in beem.discussions), 115
- Discussions\_by\_blog (class in beem.discussions), 116
- Discussions\_by\_cashout (class in beem.discussions), 116
- Discussions\_by\_children (class in beem.discussions), 116
- Discussions\_by\_comments (class in beem.discussions), 116
- Discussions\_by\_created (class in beem.discussions), 117
- Discussions\_by\_feed (class in beem.discussions), 117
- Discussions\_by\_hot (class in beem.discussions), 117
- Discussions\_by\_promoted (class in beem.discussions), 117
- Discussions\_by\_trending (class in beem.discussions), 118
- Discussions\_by\_votes (class in beem.discussions), 118
- done() (beem.blockchain.Pool method), 106
- doublesha256() (in module beemgraphenebase.base58), 164
- downvote() (beem.comment.Comment method), 108
- ## E
- edit() (beem.comment.Comment method), 108
- encode\_memo() (in module beembase.memo), 157
- encode\_memo\_bts() (in module beembase.memo), 157
- encrypt() (beem.aes.AESCipher method), 82
- encrypt() (beem.memo.Memo method), 130
- encrypt() (in module beemgraphenebase.bip38), 164
- encrypt\_token() (beem.wallet.Wallet method), 146
- encrypt\_wif() (beem.wallet.Wallet method), 146

enqueue() (beem.blockchain.Pool method), 106  
ensure\_full() (beem.account.Account method), 66  
error\_cnt (beemapi.graphenerpc.GrapheneRPC attribute), 155  
error\_cnt (beemapi.node.Nodes attribute), 154  
error\_cnt\_call (beemapi.graphenerpc.GrapheneRPC attribute), 155  
error\_cnt\_call (beemapi.node.Nodes attribute), 154  
estimate\_curation\_SBD() (beem.comment.Comment method), 108  
estimate\_virtual\_op\_num() (beem.account.Account method), 66  
ExchangeRate (class in beembase.objects), 159  
exists\_table() (beem.storage.Configuration method), 137  
exists\_table() (beem.storage.Key method), 138  
exists\_table() (beem.storage.Token method), 140  
export\_working\_nodes() (beemapi.node.Nodes method), 154  
Extension (class in beembase.objects), 159

## F

feed\_history() (beem.account.Account method), 67  
feed\_publish() (beem.witness.Witness method), 149  
FilledOrder (class in beem.price), 132  
finalizeOp() (beem.steem.Steem method), 90  
find\_rc\_accounts() (beem.blockchain.Blockchain method), 102  
findall\_patch\_hunks() (in module beem.utils), 142  
FOLLOW  
    beempy-follow command line option, 28  
follow() (beem.account.Account method), 67  
FOREIGN\_ACCOUNT  
    beempy-allow command line option, 20  
    beempy-disallow command line option, 27  
formatTime() (in module beem.utils), 142  
formatTimedelta() (in module beem.utils), 143  
formatTimeFromNow() (in module beem.utils), 142  
formatTimeString() (in module beem.utils), 143  
formatTimeStamp() (in module beem.utils), 143

## G

get() (beem.blockchainobject.ObjectCache method), 107  
get() (beem.storage.Configuration method), 137  
get\_access\_token() (beem.steemconnect.SteemConnect method), 99  
get\_account() (beemapi.steemnodeRPC.SteemNodeRPC method), 151  
get\_account\_bandwidth() (beem.account.Account method), 68  
get\_account\_count() (beem.blockchain.Blockchain method), 103  
get\_account\_history() (beem.account.Account method), 68

get\_account\_history() (beem.snapshot.AccountSnapshot method), 136  
get\_account\_reputations() (beem.blockchain.Blockchain method), 103  
get\_account\_votes() (beem.account.Account method), 68  
get\_all\_accounts() (beem.blockchain.Blockchain method), 103  
get\_all\_replies() (beem.comment.Comment method), 108  
get\_api\_methods() (beem.steem.Steem method), 90  
get\_apis() (beem.steem.Steem method), 90  
get\_author\_rewards() (beem.comment.Comment method), 108  
get\_balance() (beem.account.Account method), 68  
get\_balances() (beem.account.Account method), 69  
get\_bandwidth() (beem.account.Account method), 69  
get\_beneficiaries\_pct() (beem.comment.Comment method), 109  
get\_blind\_private() (beem-graphenebase.account.BrainKey method), 161  
get\_block\_interval() (beem.steem.Steem method), 91  
get\_blockchain\_version() (beem.steem.Steem method), 91  
get\_blog() (beem.account.Account method), 69  
get\_blog\_authors() (beem.account.Account method), 69  
get\_blog\_entries() (beem.account.Account method), 70  
get\_brainkey() (beemgraphenebase.account.BrainKey method), 161  
get\_cache\_auto\_clean() (beem.blockchainobject.BlockchainObject method), 107  
get\_cache\_expiration() (beem.blockchainobject.BlockchainObject method), 107  
get\_chain\_properties() (beem.steem.Steem method), 91  
get\_config() (beem.steem.Steem method), 91  
get\_conversion\_requests() (beem.account.Account method), 70  
get\_creator() (beem.account.Account method), 70  
get\_curation\_penalty() (beem.comment.Comment method), 109  
get\_curation\_reward() (beem.account.Account method), 70  
get\_curation\_rewards() (beem.comment.Comment method), 109  
get\_current\_block() (beem.blockchain.Blockchain method), 103  
get\_current\_block\_num() (beem.blockchain.Blockchain method), 103  
get\_current\_median\_history() (beem.steem.Steem method), 91  
get\_data() (beem.snapshot.AccountSnapshot method), 136  
get\_default\_nodes() (beem.steem.Steem method), 91  
get\_discussions() (beem.discussions.Discussions method), 114

- [get\\_dust\\_threshold\(\)](#) (beem.steem.Steem method), [91](#)  
[get\\_dynamic\\_global\\_properties\(\)](#) (beem.steem.Steem method), [91](#)  
[get\\_effective\\_vesting\\_shares\(\)](#) (beem.account.Account method), [70](#)  
[get\\_escrow\(\)](#) (beem.account.Account method), [70](#)  
[get\\_estimated\\_block\\_num\(\)](#) (beem.blockchain.Blockchain method), [103](#)  
[get\\_expiring\\_vesting\\_delegations\(\)](#) (beem.account.Account method), [71](#)  
[get\\_feature\\_flag\(\)](#) (beem.conveyor.Conveyor method), [112](#)  
[get\\_feature\\_flags\(\)](#) (beem.conveyor.Conveyor method), [112](#)  
[get\\_feed\(\)](#) (beem.account.Account method), [71](#)  
[get\\_feed\\_entries\(\)](#) (beem.account.Account method), [71](#)  
[get\\_feed\\_history\(\)](#) (beem.steem.Steem method), [91](#)  
[get\\_follow\\_count\(\)](#) (beem.account.Account method), [72](#)  
[get\\_followers\(\)](#) (beem.account.Account method), [72](#)  
[get\\_following\(\)](#) (beem.account.Account method), [72](#)  
[get\\_hardfork\\_properties\(\)](#) (beem.steem.Steem method), [91](#)  
[get\\_list\(\)](#) (beem.vote.VotesObject method), [144](#)  
[get\\_login\\_url\(\)](#) (beem.steemconnect.SteemConnect method), [99](#)  
[get\\_manabar\(\)](#) (beem.account.Account method), [72](#)  
[get\\_manabar\\_recharge\\_time\(\)](#) (beem.account.Account method), [72](#)  
[get\\_manabar\\_recharge\\_time\\_str\(\)](#) (beem.account.Account method), [72](#)  
[get\\_manabar\\_recharge\\_timedelta\(\)](#) (beem.account.Account method), [72](#)  
[get\\_median\\_price\(\)](#) (beem.steem.Steem method), [91](#)  
[get\\_muters\(\)](#) (beem.account.Account method), [72](#)  
[get\\_mutings\(\)](#) (beem.account.Account method), [72](#)  
[get\\_network\(\)](#) (beem.steem.Steem method), [91](#)  
[get\\_network\(\)](#) (beemapi.graphenerpc.GrapheneRPC method), [156](#)  
[get\\_nodes\(\)](#) (beem.nodelist.NodeList method), [97](#)  
[get\\_ops\(\)](#) (beem.snapshot.AccountSnapshot method), [136](#)  
[get\\_owner\\_history\(\)](#) (beem.account.Account method), [72](#)  
[get\\_parent\(\)](#) (beem.comment.Comment method), [109](#)  
[get\\_parent\(\)](#) (beem.transactionbuilder.TransactionBuilder method), [141](#)  
[get\\_potential\\_signatures\(\)](#) (beem.transactionbuilder.TransactionBuilder method), [141](#)  
[get\\_private\(\)](#) (beemgraphenebase.account.BrainKey method), [161](#)  
[get\\_private\(\)](#) (beemgraphenebase.account.PasswordKey method), [162](#)  
[get\\_private\\_key\(\)](#) (beemgraphenebase.account.BrainKey method), [161](#)  
[get\\_private\\_key\(\)](#) (beemgraphenebase.account.PasswordKey method), [162](#)  
[get\\_public\(\)](#) (beemgraphenebase.account.BrainKey method), [161](#)  
[get\\_public\(\)](#) (beemgraphenebase.account.PasswordKey method), [162](#)  
[get\\_public\\_key\(\)](#) (beemgraphenebase.account.Address method), [161](#)  
[get\\_public\\_key\(\)](#) (beemgraphenebase.account.BrainKey method), [161](#)  
[get\\_public\\_key\(\)](#) (beemgraphenebase.account.PasswordKey method), [162](#)  
[get\\_public\\_key\(\)](#) (beemgraphenebase.account.PrivateKey method), [162](#)  
[get\\_public\\_key\(\)](#) (beemgraphenebase.account.PublicKey method), [163](#)  
[get\\_rc\(\)](#) (beem.account.Account method), [72](#)  
[get\\_rc\\_cost\(\)](#) (beem.steem.Steem method), [91](#)  
[get\\_rc\\_manabar\(\)](#) (beem.account.Account method), [73](#)  
[get\\_reblogged\\_by\(\)](#) (beem.comment.Comment method), [109](#)  
[get\\_recharge\\_time\(\)](#) (beem.account.Account method), [73](#)  
[get\\_recharge\\_time\\_str\(\)](#) (beem.account.Account method), [73](#)  
[get\\_recharge\\_timedelta\(\)](#) (beem.account.Account method), [73](#)  
[get\\_recovery\\_request\(\)](#) (beem.account.Account method), [73](#)  
[get\\_replies\(\)](#) (beem.comment.Comment method), [109](#)  
[get\\_reputation\(\)](#) (beem.account.Account method), [73](#)  
[get\\_request\\_id\(\)](#) (beemapi.graphenerpc.GrapheneRPC method), [156](#)  
[get\\_request\\_id\(\)](#) (beemapi.websocket.SteemWebsocket method), [153](#)  
[get\\_required\\_signatures\(\)](#) (beem.transactionbuilder.TransactionBuilder method), [141](#)  
[get\\_reserve\\_ratio\(\)](#) (beem.steem.Steem method), [92](#)  
[get\\_resource\\_count\(\)](#) (beem.rc.RC method), [135](#)  
[get\\_resource\\_params\(\)](#) (beem.steem.Steem method), [92](#)  
[get\\_resource\\_pool\(\)](#) (beem.steem.Steem method), [92](#)  
[get\\_reward\\_funds\(\)](#) (beem.steem.Steem method), [92](#)  
[get\\_rewards\(\)](#) (beem.comment.Comment method), [110](#)  
[get\\_savings\\_withdrawals\(\)](#) (beem.account.Account method), [73](#)  
[get\\_sbd\\_per\\_rshares\(\)](#) (beem.steem.Steem method), [92](#)  
[get\\_secret\(\)](#) (beemgraphenebase.account.PrivateKey method), [162](#)  
[get\\_shared\\_secret\(\)](#) (in module beembase.memo), [158](#)  
[get\\_similar\\_account\\_names\(\)](#) (beem.account.Account method), [73](#)  
[get\\_similar\\_account\\_names\(\)](#)

- (beem.blockchain.Blockchain method), 103
- get\_sorted\_list() (beem.vote.VotesObject method), 144
- get\_steem\_per\_mvest() (beem.steem.Steem method), 92
- get\_steem\_power() (beem.account.Account method), 74
- get\_string() (beem.market.Market method), 124
- get\_tags\_used\_by\_author() (beem.account.Account method), 74
- get\_testnet() (beem.nodelist.NodeList method), 97
- get\_transaction() (beem.blockchain.Blockchain method), 104
- get\_transaction\_hex() (beem.blockchain.Blockchain method), 104
- get\_transaction\_hex() (beem.transactionbuilder.TransactionBuilder method), 141
- get\_tx\_size() (beem.rc.RC method), 135
- get\_use\_appbase() (beemapi.graphenerpc.GrapheneRPC method), 156
- get\_user\_data() (beem.conveyor.Conveyor method), 112
- get\_vesting\_delegations() (beem.account.Account method), 74
- get\_vests() (beem.account.Account method), 74
- get\_vote() (beem.account.Account method), 74
- get\_vote\_pct\_for\_SBD() (beem.account.Account method), 74
- get\_vote\_with\_curation() (beem.comment.Comment method), 110
- get\_votes() (beem.comment.Comment method), 110
- get\_votes\_sum() (beem.witness.WitnessesObject method), 150
- get\_voting\_power() (beem.account.Account method), 74
- get\_voting\_value\_SBD() (beem.account.Account method), 74
- get\_withdraw\_routes() (beem.account.Account method), 74
- get\_witness\_schedule() (beem.steem.Steem method), 92
- getAccount() (beem.wallet.Wallet method), 146
- getAccountFromPrivateKey() (beem.wallet.Wallet method), 146
- getAccountFromPublicKey() (beem.wallet.Wallet method), 146
- getAccounts() (beem.wallet.Wallet method), 146
- getAccountsFromPublicKey() (beem.wallet.Wallet method), 146
- getActiveKeyForAccount() (beem.wallet.Wallet method), 146
- getActiveKeysForAccount() (beem.wallet.Wallet method), 146
- getAllAccounts() (beem.wallet.Wallet method), 146
- getBlockParams() (in module beembase.transactions), 160
- getcache() (beem.blockchainobject.BlockchainObject method), 107
- getChainParams() (beem-graphenebase.signedtransactions.Signed\_Transaction method), 166
- getEncryptedMaster() (beem.storage.MasterPassword method), 139
- getKeyForAccount() (beem.wallet.Wallet method), 147
- getKeysForAccount() (beem.wallet.Wallet method), 147
- getKeyType() (beem.wallet.Wallet method), 147
- getKnownChains() (beem-base.signedtransactions.Signed\_Transaction method), 160
- getKnownChains() (beem-graphenebase.signedtransactions.Signed\_Transaction method), 166
- getMemoKeyForAccount() (beem.wallet.Wallet method), 147
- getOperationClass() (beem-base.signedtransactions.Signed\_Transaction method), 160
- getOperationClass() (beem-graphenebase.signedtransactions.Signed\_Transaction method), 166
- getOperationNameForId() (beembase.objects.Operation method), 159
- getOperationNameForId() (beem-graphenebase.objects.Operation method), 165
- getOperationNameForId() (in module beem-base.operationids), 159
- getOwnerKeyForAccount() (beem.wallet.Wallet method), 147
- getOwnerKeysForAccount() (beem.wallet.Wallet method), 147
- getPostingKeyForAccount() (beem.wallet.Wallet method), 147
- getPostingKeysForAccount() (beem.wallet.Wallet method), 147
- getPrivateKeyForPublicKey() (beem.storage.Key method), 138
- getPrivateKeyForPublicKey() (beem.wallet.Wallet method), 147
- getPublicKeys() (beem.storage.Key method), 138
- getPublicKeys() (beem.wallet.Wallet method), 147
- getPublicNames() (beem.storage.Token method), 140
- getPublicNames() (beem.wallet.Wallet method), 147
- getSimilarAccountNames() (beem.account.Account method), 68
- getTokenForAccountName() (beem.wallet.Wallet method), 147
- getTokenForPublicName() (beem.storage.Token method), 140
- gphBase58CheckDecode() (in module beem-graphenebase.base58), 164
- gphBase58CheckEncode() (in module beem-graphenebase.base58), 164
- GrapheneObject (class in beemgraphenebase.objects), 164

165  
 GrapheneRPC (class in beemapi.graphenerpc), 155

## H

hardfork (beem.steem.Steem attribute), 92  
 has\_voted() (beem.account.Account method), 75  
 hash\_op() (beem.blockchain.Blockchain static method), 104  
 headers (beem.steemconnect.SteemConnect attribute), 99  
 healthcheck() (beem.conveyor.Conveyor method), 113  
 history() (beem.account.Account method), 75  
 history\_reverse() (beem.account.Account method), 76

## I

id (beem.comment.Comment attribute), 110  
 id (beemgraphenebase.signedtransactions.Signed\_Transaction attribute), 166

### IDENTIFIER

beem.py-resteem command line option, 36  
 idle() (beem.blockchain.Pool method), 106  
 ImageUploader (class in beem.imageuploader), 122  
 increase\_error\_cnt() (beemapi.node.Nodes method), 154  
 increase\_error\_cnt\_call() (beemapi.node.Nodes method), 154  
 info() (beem.steem.Steem method), 92  
 init\_aes() (in module beembase.memo), 158  
 init\_aes\_bts() (in module beembase.memo), 158  
 instance (beem.instance.SharedInstance attribute), 122  
 instance (beemapi.graphenerpc.SessionInstance attribute), 156  
 InsufficientAuthorityError, 120  
 interest() (beem.account.Account method), 77  
 InvalidAssetException, 120  
 InvalidEndpointUrl, 151  
 InvalidMemoKeyException, 121  
 InvalidMessageSignature, 121  
 InvalidWifiError, 121  
 invert() (beem.price.Price method), 133  
 is\_active (beem.witness.Witness attribute), 149  
 is\_appbase\_ready() (beemapi.graphenerpc.GrapheneRPC method), 156  
 is\_comment() (beem.comment.Comment method), 110  
 is\_connected() (beem.steem.Steem method), 92  
 is\_empty() (beem.transactionbuilder.TransactionBuilder method), 141  
 is\_fully\_loaded (beem.account.Account attribute), 77  
 is\_irreversible\_mode() (beem.blockchain.Blockchain method), 104  
 is\_main\_post() (beem.comment.Comment method), 110  
 is\_pending() (beem.comment.Comment method), 110  
 isArgsThisClass() (in module beemgraphenebase.objects), 165  
 iscached() (beem.blockchainobject.BlockchainObject method), 107

items() (beem.blockchainobject.BlockchainObject method), 107  
 items() (beem.storage.Configuration method), 137

## J

join() (beem.blockchain.Pool method), 106  
 json() (beem.account.Account method), 77  
 json() (beem.amount.Amount method), 84  
 json() (beem.block.Block method), 100  
 json() (beem.block.BlockHeader method), 101  
 json() (beem.blockchainobject.BlockchainObject method), 107  
 json() (beem.comment.Comment method), 110  
 json() (beem.price.FilledOrder method), 132  
 json() (beem.price.Price method), 134  
 json() (beem.transactionbuilder.TransactionBuilder method), 141  
 json() (beem.vote.Vote method), 144  
 json() (beem.witness.Witness method), 149  
 json() (beembase.objects.Operation method), 159  
 json() (beemgraphenebase.objects.GrapheneObject method), 165  
 json\_metadata (beem.account.Account attribute), 77  
 json\_metadata (beem.comment.Comment attribute), 110  
 json\_operations (beem.block.Block attribute), 100  
 json\_transactions (beem.block.Block attribute), 100

## K

KEY  
 beem.py-set command line option, 38  
 Key (class in beem.storage), 138  
 keyMap (beem.wallet.Wallet attribute), 147  
 keys (beem.wallet.Wallet attribute), 147  
 keyStorage (beem.wallet.Wallet attribute), 147

## L

list\_drafts() (beem.conveyor.Conveyor method), 113  
 list\_operations() (beem.transactionbuilder.TransactionBuilder method), 141  
 listen() (beem.notify.Notify method), 131  
 ListWitnesses (class in beem.witness), 148  
 lock() (beem.wallet.Wallet method), 147  
 locked() (beem.wallet.Wallet method), 148  
 log (in module beemgraphenebase.base58), 164

## M

make\_patch() (in module beem.utils), 143  
 market (beem.price.Price attribute), 134  
 Market (class in beem.market), 123  
 market\_history() (beem.market.Market method), 124  
 market\_history\_buckets() (beem.market.Market method), 125  
 MasterPassword (beem.wallet.Wallet attribute), 145



masterpassword (beem.wallet.Wallet attribute), 148  
MasterPassword (class in beem.storage), 139  
me() (beem.steemconnect.SteemConnect method), 99  
MEMO  
    beempy-transfer command line option, 40  
Memo (class in beem.memo), 128  
Memo (class in beembase.objects), 159  
Message (class in beem.message), 131  
MissingKeyError, 121  
MissingRequiredActiveAuthority, 151  
mkdir\_p() (beem.storage.DataDir method), 138  
move\_current\_node\_to\_front() (beem.steem.Steem method), 92  
MUTE  
    beempy-mute command line option, 31  
mute() (beem.account.Account method), 77

## N

### NAME

    beempy-addtoken command line option, 19  
    beempy-deltoken command line option, 26  
name (beem.account.Account attribute), 77  
new\_chart() (beem.asciichart.AsciiChart method), 83  
new\_tx() (beem.steem.Steem method), 92  
newMaster() (beem.storage.MasterPassword method), 139  
newWallet() (beem.steem.Steem method), 92  
newWallet() (beem.wallet.Wallet method), 148  
next() (beemapi.graphenerpc.GrapheneRPC method), 156  
next() (beemapi.node.Nodes method), 154  
next\_sequence() (beemgraphenebase.account.BrainKey method), 161  
NoAccessApi, 151  
NoApiWithName, 151  
node (beemapi.node.Nodes attribute), 154  
Node (class in beemapi.node), 154  
nodelist (beem.storage.Configuration attribute), 137  
NodeList (class in beem.nodelist), 97  
nodes (beem.storage.Configuration attribute), 137  
Nodes (class in beemapi.node), 154  
NoMethodWithName, 151  
normalize() (beemgraphenebase.account.BrainKey method), 161  
Notify (class in beem.notify), 131  
NoWalletException, 121  
NoWriteAccess, 121  
num\_retries (beemapi.graphenerpc.GrapheneRPC attribute), 156  
num\_retries\_call (beemapi.graphenerpc.GrapheneRPC attribute), 156  
num\_retries\_call\_reached (beemapi.node.Nodes attribute), 154  
NumRetriesReached, 151

## O

object\_type (in module beembase.objecttypes), 159  
object\_type (in module beemgraphenebase.objecttypes), 165  
ObjectCache (class in beem.blockchainobject), 107  
OBJECTS  
    beempy-info command line option, 29  
OfflineHasNoRPCEException, 121  
on\_close() (beemapi.websocket.SteemWebsocket method), 153  
on\_error() (beemapi.websocket.SteemWebsocket method), 153  
on\_message() (beemapi.websocket.SteemWebsocket method), 153  
on\_open() (beemapi.websocket.SteemWebsocket method), 153  
Operation (class in beembase.objects), 159  
Operation (class in beemgraphenebase.objects), 165  
operations (beem.block.Block attribute), 100  
operations (in module beemgraphenebase.operationids), 165  
operations() (beembase.objects.Operation method), 159  
operations() (beemgraphenebase.objects.Operation method), 165  
ops (in module beembase.operationids), 159  
ops() (beem.blockchain.Blockchain method), 104  
ops\_statistics() (beem.block.Block method), 100  
ops\_statistics() (beem.blockchain.Blockchain method), 104  
Order (class in beem.price), 132  
orderbook() (beem.market.Market method), 125  
ORDERID  
    beempy-cancel command line option, 22

## P

parent\_author (beem.comment.Comment attribute), 110  
parent\_permlink (beem.comment.Comment attribute), 110  
parse\_op() (beem.snapshot.AccountSnapshot method), 136  
parse\_time() (in module beem.utils), 143  
password (beem.storage.MasterPassword attribute), 139  
PasswordKey (class in beemgraphenebase.account), 161  
percent (beem.vote.Vote attribute), 144  
Permission (class in beembase.objects), 159  
permlink (beem.comment.Comment attribute), 110  
plot() (beem.asciichart.AsciiChart method), 83  
point() (beemgraphenebase.account.PublicKey method), 163  
Pool (class in beem.blockchain), 106  
POST  
    beempy-downvote command line option, 27  
    beempy-upvote command line option, 42  
post() (beem.steem.Steem method), 92

- Post\_discussions\_by\_payout (class in beem.discussions), 118
- precision (beem.asset.Asset attribute), 85
- prefix (beem.steem.Steem attribute), 93
- prefix (beem.wallet.Wallet attribute), 148
- prehash\_message() (beem.conveyor.Conveyor method), 113
- PRICE
- beem-py-buy command line option, 21
  - beem-py-sell command line option, 38
- Price (class in beem.price), 132
- Price (class in beembase.objects), 159
- print\_info() (beem.account.Account method), 77
- print\_stats() (beem.vote.VotesObject method), 144
- print\_summarize\_table() (beem.account.AccountsObject method), 81
- printAsTable() (beem.account.AccountsObject method), 81
- printAsTable() (beem.vote.VotesObject method), 144
- printAsTable() (beem.witness.WitnessesObject method), 150
- PrivateKey (class in beemgraphenebase.account), 162
- process\_block() (beem.notify.Notify method), 131
- process\_block() (beemapi.websocket.SteemWebsocket method), 153
- profile (beem.account.Account attribute), 78
- PUB
- beem-py-delkey command line option, 25
- PUB\_SIGNING\_KEY
- beem-py-witnesscreate command line option, 44
- PublicKey (class in beemgraphenebase.account), 162
- ## Q
- Query (class in beem.discussions), 119
- ## R
- RC (class in beem.rc), 134
- recent\_trades() (beem.market.Market method), 126
- RecentByPath (class in beem.comment), 111
- RecentReplies (class in beem.comment), 111
- recover\_public\_key() (in module beem-graphenebase.ecdsasig), 164
- recover\_with\_latest\_backup() (beem.storage.DataDir method), 138
- recoverPubkeyParameter() (in module beem-graphenebase.ecdsasig), 164
- refresh() (beem.account.Account method), 78
- refresh() (beem.asset.Asset method), 85
- refresh() (beem.block.Block method), 100
- refresh() (beem.block.BlockHeader method), 101
- refresh() (beem.comment.Comment method), 110
- refresh() (beem.vote.Vote method), 144
- refresh() (beem.witness.Witness method), 149
- refresh() (beem.witness.Witnesses method), 150
- refresh\_access\_token() (beem.steemconnect.SteemConnect method), 99
- refresh\_data() (beem.steem.Steem method), 94
- refreshBackup() (beem.storage.DataDir method), 138
- remove\_draft() (beem.conveyor.Conveyor method), 113
- remove\_from\_dict() (in module beem.utils), 143
- removeAccount() (beem.wallet.Wallet method), 148
- removePrivateKeyFromPublicKey() (beem.wallet.Wallet method), 148
- removeTokenFromPublicName() (beem.wallet.Wallet method), 148
- rep (beem.account.Account attribute), 78
- rep (beem.vote.Vote attribute), 144
- Replies\_by\_last\_update (class in beem.discussions), 119
- reply() (beem.comment.Comment method), 110
- reply\_history() (beem.account.Account method), 78
- reputation (beem.vote.Vote attribute), 144
- reputation\_to\_score() (in module beem.utils), 143
- request\_send() (beemapi.graphenerpc.GrapheneRPC method), 156
- reset() (beem.snapshot.AccountSnapshot method), 136
- reset\_error\_cnt() (beemapi.node.Nodes method), 154
- reset\_error\_cnt\_call() (beemapi.node.Nodes method), 154
- reset\_subscriptions() (beem.notify.Notify method), 131
- reset\_subscriptions() (beemapi.websocket.SteemWebsocket method), 153
- resolve\_authorperm() (in module beem.utils), 143
- resolve\_authorpermvoter() (in module beem.utils), 143
- resolve\_root\_identifier() (in module beem.utils), 143
- resteem() (beem.comment.Comment method), 110
- results() (beem.blockchain.Pool method), 106
- revoke\_token() (beem.steemconnect.SteemConnect method), 99
- reward (beem.comment.Comment attribute), 111
- reward\_balances (beem.account.Account attribute), 78
- ripemd160() (in module beemgraphenebase.base58), 164
- rpc (beem.wallet.Wallet attribute), 148
- rpcclose() (beemapi.graphenerpc.GrapheneRPC method), 156
- rpcconnect() (beemapi.graphenerpc.GrapheneRPC method), 156
- RPCCConnection, 151
- RPCCConnectionRequired, 121
- RPCError, 152
- RPCErrorDoRetry, 152
- rpcexec() (beemapi.graphenerpc.GrapheneRPC method), 156
- rpcexec() (beemapi.steemnoderpc.SteemNodeRPC method), 151
- rpcexec() (beemapi.websocket.SteemWebsocket method), 154
- rpclogin() (beemapi.graphenerpc.GrapheneRPC method), 156

rshares (beem.vote.Vote attribute), 144  
rshares\_to\_sbd() (beem.steem.Steem method), 94  
rshares\_to\_vote\_pct() (beem.steem.Steem method), 94  
run() (beem.blockchain.Pool method), 106  
run() (beem.blockchain.Worker method), 106  
run\_forever() (beemapi.websocket.SteemWebsocket method), 154

## S

SaltException, 164  
sanitize\_permalink() (in module beem.utils), 143  
save\_draft() (beem.conveyor.Conveyor method), 114  
saveEncryptedMaster() (beem.storage.MasterPassword method), 139  
saving\_balances (beem.account.Account attribute), 78  
sbd (beem.vote.Vote attribute), 144  
sbd\_symbol (beem.steem.Steem attribute), 94  
sbd\_to\_rshares() (beem.steem.Steem method), 94  
sbd\_to\_vote\_pct() (beem.steem.Steem method), 94  
search() (beem.snapshot.AccountSnapshot method), 136  
sell() (beem.market.Market method), 127  
SessionInstance (class in beemapi.graphenerpc), 156  
set\_access\_token() (beem.steemconnect.SteemConnect method), 99  
set\_cache\_auto\_clean() (beem.blockchainobject.BlockchainObject method), 107  
set\_cache\_expiration() (beem.blockchainobject.BlockchainObject method), 107  
set\_default\_account() (beem.steem.Steem method), 95  
set\_default\_nodes() (beem.steem.Steem method), 95  
set\_default\_vote\_weight() (beem.steem.Steem method), 95  
set\_expiration() (beem.transactionbuilder.TransactionBuilder method), 142  
set\_next\_node\_on\_empty\_reply() (beemapi.steemnodeRPC.SteemNodeRPC method), 151  
set\_parameter() (beem.asciichart.AsciiChart method), 83  
set\_password\_storage() (beem.steem.Steem method), 95  
set\_session\_instance() (in module beemapi.graphenerpc), 156  
set\_shared\_config() (in module beem.instance), 122  
set\_shared\_steem\_instance() (in module beem.instance), 123  
set\_user\_data() (beem.conveyor.Conveyor method), 114  
set\_username() (beem.steemconnect.SteemConnect method), 99  
set\_withdraw\_vesting\_route() (beem.account.Account method), 78  
setKeys() (beem.wallet.Wallet method), 148  
setToken() (beem.wallet.Wallet method), 148  
shared\_session\_instance() (in module beemapi.graphenerpc), 156  
shared\_steem\_instance() (in module beem.instance), 123

SharedInstance (class in beem.instance), 122  
sign() (beem.message.Message method), 131  
sign() (beem.steem.Steem method), 95  
sign() (beem.transactionbuilder.TransactionBuilder method), 142  
sign() (beembase.signedtransactions.Signed\_Transaction method), 160  
sign() (beemgraphenebase.signedtransactions.Signed\_Transaction method), 166  
sign\_message() (in module beemgraphenebase.ecdsasig), 164  
Signed\_Transaction (class in beembase.signedtransactions), 160  
Signed\_Transaction (class in beemgraphenebase.signedtransactions), 165  
SIGNING\_KEY  
    beempy-witnessenable command line option, 44  
sleep\_and\_check\_retries() (beemapi.node.Nodes method), 154  
sp (beem.account.Account attribute), 78  
sp\_to\_rshares() (beem.steem.Steem method), 95  
sp\_to\_sbd() (beem.steem.Steem method), 95  
sp\_to\_vests() (beem.steem.Steem method), 95  
space\_id (beem.blockchainobject.BlockchainObject attribute), 107  
sqlDataBaseFile (beem.storage.DataDir attribute), 138  
sqlite3\_backup() (beem.storage.DataDir method), 138  
sqlite3\_copy() (beem.storage.DataDir method), 138  
Steem (class in beem.steem), 85  
steem\_btc\_ticker() (beem.market.Market static method), 127  
steem\_symbol (beem.steem.Steem attribute), 95  
steem\_usd\_implied() (beem.market.Market method), 127  
SteemConnect (class in beem.steemconnect), 98  
SteemNodeRPC (class in beemapi.steemnodeRPC), 150  
SteemWebsocket (class in beemapi.websocket), 152  
stop() (beemapi.websocket.SteemWebsocket method), 154  
storageDatabase (beem.storage.DataDir attribute), 138  
str\_to\_bytes() (beem.aes.AESCipher static method), 82  
stream() (beem.blockchain.Blockchain method), 104  
suggest() (beemgraphenebase.account.BrainKey method), 161  
symbol (beem.amount.Amount attribute), 84  
symbol (beem.asset.Asset attribute), 85  
symbols() (beem.price.Price method), 134

## T

test\_valid\_objectid() (beem.blockchainobject.BlockchainObject method), 107  
testid() (beem.blockchainobject.BlockchainObject method), 107  
ticker() (beem.market.Market method), 127  
time (beem.vote.Vote attribute), 144



- time() (beem.block.Block method), 100  
time() (beem.block.BlockHeader method), 101  
time\_elapsed() (beem.comment.Comment method), 111  
TimeoutException, 152  
title (beem.comment.Comment attribute), 111  
TO  
    beem-powerroute command line option, 35  
    beem-transfer command line option, 40  
TO\_ACCOUNT  
    beem-delegate command line option, 25  
toJson() (beemgraphenebase.objects.GrapheneObject method), 165  
token (beem.wallet.Wallet attribute), 148  
Token (class in beem.storage), 139  
tokenStorage (beem.wallet.Wallet attribute), 148  
total\_balances (beem.account.Account attribute), 78  
trade\_history() (beem.market.Market method), 128  
trades() (beem.market.Market method), 128  
TransactionBuilder (class in beem.transactionbuilder), 140  
transactions (beem.block.Block attribute), 100  
transfer() (beem.account.Account method), 79  
transfer() (beem.rc.RC method), 135  
transfer\_dict() (beem.rc.RC method), 135  
transfer\_from\_savings() (beem.account.Account method), 79  
transfer\_to\_savings() (beem.account.Account method), 79  
transfer\_to\_vesting() (beem.account.Account method), 79  
Trending\_tags (class in beem.discussions), 120  
tryUnlockFromEnv() (beem.wallet.Wallet method), 148  
tuple() (beem.amount.Amount method), 84  
tx() (beem.steem.Steem method), 95  
txbuffer (beem.steem.Steem attribute), 96  
type\_id (beem.account.Account attribute), 80  
type\_id (beem.asset.Asset attribute), 85  
type\_id (beem.blockchainobject.BlockchainObject attribute), 107  
type\_id (beem.comment.Comment attribute), 111  
type\_id (beem.vote.Vote attribute), 144  
type\_id (beem.witness.Witness attribute), 149  
type\_ids (beem.blockchainobject.BlockchainObject attribute), 107
- ## U
- UnauthorizedError, 152  
unCompressed() (beemgraphenebase.account.PublicKey method), 163  
UNFOLLOW  
    beem-unfollow command line option, 40  
unfollow() (beem.account.Account method), 80  
UnhandledRPCError, 152  
UnkownKey, 152  
unlock() (beem.steem.Steem method), 96  
unlock() (beem.wallet.Wallet method), 148  
unlock\_wallet() (beem.memo.Memo method), 130  
unlocked() (beem.wallet.Wallet method), 148  
UnnecessarySignatureDetected, 152  
update() (beem.snapshot.AccountSnapshot method), 136  
update() (beem.witness.Witness method), 149  
update\_account\_metadata() (beem.account.Account method), 80  
update\_account\_profile() (beem.account.Account method), 80  
update\_in\_vote() (beem.snapshot.AccountSnapshot method), 137  
update\_memo\_key() (beem.account.Account method), 80  
update\_nodes() (beem.nodelist.NodeList method), 97  
update\_out\_vote() (beem.snapshot.AccountSnapshot method), 137  
update\_rewards() (beem.snapshot.AccountSnapshot method), 137  
update\_user\_metadata() (beem.steemconnect.SteemConnect method), 99  
updateToken() (beem.storage.Token method), 140  
updateWif() (beem.storage.Key method), 138  
upload() (beem.imageuploader.ImageUploader method), 122  
upvote() (beem.comment.Comment method), 111  
url (beemapi.node.Nodes attribute), 154  
url\_from\_tx() (beem.steemconnect.SteemConnect method), 99
- ## V
- VALUE  
    beem-set command line option, 38  
    beem-setprofile command line option, 38  
VARIABLE  
    beem-delpoof command line option, 26  
    beem-setprofile command line option, 38  
verify() (beem.message.Message method), 131  
verify() (beembase.signedtransactions.Signed\_Transaction method), 160  
verify() (beemgraphenebase.signedtransactions.Signed\_Transaction method), 166  
verify\_account\_authority() (beem.account.Account method), 81  
verify\_authority() (beem.transactionbuilder.TransactionBuilder method), 142  
verify\_message() (in module beem-graphenebase.ecdsasig), 165  
version\_string\_to\_int() (beemapi.graphenerpc.GrapheneRPC method), 156  
VestingBalanceDoesNotExistsException, 121  
vests\_symbol (beem.steem.Steem attribute), 96  
vests\_to\_rshares() (beem.steem.Steem method), 96  
vests\_to\_sbd() (beem.steem.Steem method), 96

vests\_to\_sp() (beem.steem.Steem method), 96  
virtual\_op\_count() (beem.account.Account method), 81  
volume24h() (beem.market.Market method), 128  
Vote (class in beem.vote), 144  
vote() (beem.comment.Comment method), 111  
vote() (beem.rc.RC method), 135  
vote\_dict() (beem.rc.RC method), 135  
VOTE\_WEIGHT  
    beempy-downvote command line option, 27  
    beempy-upvote command line option, 42  
VoteDoesNotExistsException, 121  
votee (beem.vote.Vote attribute), 144  
voter (beem.vote.Vote attribute), 144  
VotesObject (class in beem.vote), 144  
VotingInvalidOnArchivedPost, 121  
vp (beem.account.Account attribute), 81

## W

wait\_for\_and\_get\_block() (beem.blockchain.Blockchain method), 105  
Wallet (class in beem.wallet), 145  
WalletExists, 121  
WalletLocked, 121  
weight (beem.vote.Vote attribute), 144  
WIF  
    beempy-witnessfeed command line option, 45  
    beempy-witnessproperties command line option, 46  
wipe() (beem.storage.Key method), 139  
wipe() (beem.storage.MasterPassword static method), 139  
wipe() (beem.storage.Token method), 140  
wipe() (beem.wallet.Wallet method), 148  
withdraw\_vesting() (beem.account.Account method), 81  
WITNESS  
    beempy-approvewitness command line option, 20  
    beempy-disapprovewitness command line option, 27  
    beempy-witness command line option, 43  
    beempy-witnesscreate command line option, 44  
    beempy-witnessdisable command line option, 44  
    beempy-witnessenable command line option, 44  
    beempy-witnessfeed command line option, 45  
    beempy-witnessproperties command line option, 46  
Witness (class in beem.witness), 149  
witness\_set\_properties() (beem.steem.Steem method), 96  
witness\_update() (beem.steem.Steem method), 97  
WitnessDoesNotExistsException, 122  
Witnesses (class in beem.witness), 149  
WitnessesObject (class in beem.witness), 150  
WitnessesRankedByVote (class in beem.witness), 150  
WitnessesVotedByAccount (class in beem.witness), 150  
WitnessProps (class in beembase.objects), 159  
Worker (class in beem.blockchain), 106  
working\_nodes\_count (beemapi.node.Nodes attribute), 154

WorkingNodeMissing, 152  
WrongMasterPasswordException, 122  
WrongMemoKey, 122  
ws\_send() (beemapi.graphenerpc.GrapheneRPC method), 156