
beem Documentation

Release 0.19.45

Holger Nahrstaedt

Jul 10, 2018

Contents

1	About this Library	3
2	Quickstart	5
3	General	7
4	Indices and tables	151
	Python Module Index	153

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
```

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
```

3.2 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()
```

(continues on next page)

(continued from previous page)

```
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.ops():
    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/ffdhg-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)
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 = "5KQwrPbwdL6PhXujxW37FSSQZlJiwsST4cqQzDeyXtP79zkvFD3"

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
from beem.instance import set_shared_steem_instance

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

# 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 global 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 = "5KQwrPbwdL6PhXujxW37FSSQZ1JiwsST4cqQzDeyXtP79zkvFD3"

#
# Instanciate 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.
```

(continues on next page)

(continued from previous page)

```
# 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()

if __name__ == "__main__":
    #
    # Instanciate 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
    )
```

(continues on next page)

(continued from previous page)

```
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 sendd 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"]):
    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
```

(continues on next page)

(continued from previous page)

```
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)
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

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

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

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

```
beem permissions [OPTIONS] [ACCOUNT]
```

Arguments

ACCOUNT
Optional argument

pingnode

Returns the answer time in milliseconds

```
beem 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

resteed

Resteed an existing post

```
beempy resteed [OPTIONS] IDENTIFIER
```

Options

-a, --account <account>

Resteed 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

```
beem.py 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

```
beem.py 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 updatememokey 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

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
```

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

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...
```

(continues on next page)

(continued from previous page)

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
muter	Get information about muter
muting	Get information about muting
newaccount	Create a new account
nextnode	Uses the next node in list
openorders	Show open 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 in milliseconds
power	Shows vote power and bandwidth
powerdown	Power down (start withdrawing VESTS from...
powerdownroute	Setup a powerdown route
powerup	Power up (vest STEEM as STEEM POWER)
pricehistory	Show price history
resteen	Resteen an existing post
sell	Sell STEEM or SBD from the internal market...
set	Set default_account, default_vote_weight or...
setprofile	Set a variable in an account's profile
sign	Sign a provided transaction with available...
ticker	Show ticker
tradehistory	Show price history
transfer	Transfer SBD/STEEM
unfollow	Unfollow/Unmute another account
updatememokey	Update an account's memo key
upvote	Upvote a post/comment POST is...
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 be tested by beem.py with

```
beem.py 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

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 STEEM'*, *reward_sbd='0 SBD'*, *reward_vests='0 VESTS'*, *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.

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)
```

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

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("test")
>>> 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]
```

(continues on next page)

(continued from previous page)

```
>>> print("%s - %s - %s - %s" % (entry["author"], entry["permlink"], 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("test")
>>> 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_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("test")
>>> 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("test")
>>> 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*)

Returns the account followers as list

get_following (*raw_name_list=True*)

Returns who the account is following as list

get_muters (*raw_name_list=True*)

Returns the account muters as list

get_mutings (*raw_name_list=True*)

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("test")
>>> account.get_owner_history()
[]
```

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("test")
>>> 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("test")
>>> 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("test")
>>> 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("test")
>>> account.get_vesting_delegations()
[]
```

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_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*)

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("test")
>>> 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 authorpermalink

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):
    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):
```

(continues on next page)

(continued from previous page)

```
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

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://c1.staticflickr.com/5/4715/38733717165_
↪7070227c89_n.jpg',
  'cover_image': 'https://farm1.staticflickr.com/894/26382750057_69f5c8e568.
↪jpg',
```

(continues on next page)

(continued from previous page)

```

    'website': 'https://github.com/holgern/beem'
}

```

```

from beem.account import Account
acc = Account("test")
profile = acc.profile
profile["about"] = "test account"
acc.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 RPC

```
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()
```

(continues on next page)

(continued from previous page)

```
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=False,
                           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.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 NumRetriesReached 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

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.

broadcast (*tx=None*)

Broadcast a transaction to the Steem network

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

chain_params

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, additional_owner_keys=[], additional_active_keys=[], additional_posting_keys=[], additional_owner_accounts=[], additional_active_accounts=[], additional_posting_accounts=[], storekeys=True, store_owner_key=False, json_meta=None, delegation_fee_steem='0 STEEM', **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! **You can partially pay that fee by delegating VESTS.** To pay the fee in full in STEEM, leave delegation_fee_steem set to 0 STEEM (Default). To pay the fee partially in STEEM, partially with delegated VESTS, set delegation_fee_steem to a value greater than 1 STEEM. *Required VESTS will be calculated automatically.* To pay the fee with maximum amount of delegation, set delegation_fee_steem to 1 STEEM. *Required VESTS will be calculated automatically.*

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)
- **delegation_fee_steem** – If set, *creator* pays a fee of this amount, and delegates the rest with VESTS (calculated automatically). Minimum: 1 STEEM. If left to 0 (Default), full fee is paid without VESTS delegation.
- **creator** (*str*) – which account should pay the registration fee (defaults to default_account)

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

custom_json (*id*, *json_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

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_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, replace_steemit_by_steem=False*)

Returns internal chain configuration.

Parameters

- **use_stored_data** (*bool*) – If True, the cached value is returned
- **replace_steemit_by_steem** (*bool*) – If True, it replaces all STEEMIT keys by STEEM (only useful on non appbase nodes)

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_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_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_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 (*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

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*, *permlink=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, 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, the given rshares are too high

Returns the voting participation (100% = 10000)

Parameters

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

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, 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)

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

tx()

Returns the default transaction buffer

txbuffer

Returns the currently active tx buffer

unlock (**args, **kwargs*)

Unlock the internal wallet

vests_to_rshares (*vests, voting_power=10000, vote_pct=10000, 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, *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)

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_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": x,
    "maximum_block_size": x,
    "sbd_interest_rate": x,
}
```

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, *wss*=True, *https*=True)

Returns nodes as list

Parameters

- **normal** (*bool*) – when True, nodes with version 0.19.2 or 0.19.3 are included

- **appbase** (*bool*) – when True, nodes with version 0.19.4 are included

- **dev** (*bool*) – when True, dev nodes with version 0.19.4 are included

- **testnet** (*bool*) – when True, testnet nodes are included

get_testnet()

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 steamconnect 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://v2.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()
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))
```

```
'https://v2.steemconnect.com/sign/transfer?from=test&to=test1&amount=1.000+STEEM&
↳memo=test'
```

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

Broadcast a operations

Sample operations:

```
[
  [
    '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

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 content 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.

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)
```

(continues on next page)

(continued from previous page)

```
>>> ret == ['test', 'test-1', 'test-2', 'test-ico', 'test-ilionx-123']
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: .. code-block:: js

```
{ 'type': 'transfer', 'from': 'johngreenfield', 'to': 'thundercurator', 'amount': '0.080
  SBD', 'memo': 'https://steemit.com/lofi/@johngreenfield/lofi-joji-yeah-right',
  '_id': '6d4c5f2d4d8ef1918acae4a8dce34f9da384786', 'timestamp': date-
    time.datetime(2018, 5, 9, 11, 23, 6, tzinfo=<UTC>), 'block_num': 22277588, 'trx_id':
    'cf11b2ac8493c71063ec121b2e8517ab1e0e6bea'
}
```

output when `raw_ops=True` is set: .. code-block:: js

```
{ '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)

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`)

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 30 minutes which will compensate the curation penalty, if voting earlier than 30 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.

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_author_rewards ()

Returns the author rewards.

Example::

```
{
  'pending_rewards': True,
  'payout_SP': 0.912 STEEM,
  'payout_SBD': 3.583 SBD,
  '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,
→ 'qurator-tier-1-2': 0.012 STEEM
  }
}
```

get_reblogged_by (*identifier=None*)

Shows in which blogs this post appears

get_replies (*raw_data=False, identifier=None*)

Returns all 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_permlink

permlink

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)

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

Resteem a post

Parameters

- **identifier** (*str*) – post identifier (@<account>/<permlink>)
- **account** (*str*) – (optional) the account to allow access to (defaults to `default_account`)

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.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 accesing 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_permalink="",
                                                          before_date='1970-01-01T00:00:00',
                                                          limit=100, lazy=False,
                                                          steem_instance=None)

```

Bases: list

Get Discussions by author before date

Parameters

- **discussion_query** (`beem.discussions.Query`) – Defines the parameter for searching posts
- **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=[],    se-
                              lect_authors=[],    select_tags=[],    start_author=None,
                              start_permlink=None,    start_tag=None,    parent_author=None,
                              parent_permlink=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) -`

```
from beem.discussions import Query
query = Query(limit=10, tag="steemit")
```

class `beem.discussions.Replies_by_last_update` (*discussion_query*, *lazy=False*, *steem_instance=None*)

Bases: `list`

Returns a list of replies by last update

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, Replies_by_last_update
q = Query(limit=10, start_author="steemit", start_permlink="firstpost")
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.AccountDoesNotExistsException`

Bases: `Exception`

The account does not exist

exception `beem.exceptions.AccountExistsException`

Bases: `Exception`

The requested account already exists

exception `beem.exceptions.AssetDoesNotExistsException`

Bases: `Exception`

The asset does not exist

exception `beem.exceptions.BatchedCallsNotSupported`

Bases: `Exception`

Batch calls do not work

exception `beem.exceptions.BlockDoesNotExistsException`

Bases: `Exception`

The block does not exist

exception `beem.exceptions.BlockWaitTimeExceeded`

Bases: `Exception`

Wait time for new block exceeded

exception `beem.exceptions.ContentDoesNotExistsException`

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.OfflineHasNoRPCException`

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(steam_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(*steam_instance*)

This method allows us to override default steem instance for all users of `SharedInstance.instance`.

Parameters **steam_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", steam_instance=shared_steem_instance())
```

beem.market

class beem.market.Market (*base=None, quote=None, steam_instance=None*)

Bases: dict

This class allows to easily access Markets on the blockchain for trading, etc.

Parameters

- **steam_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 *returnMarketHistory-Buckets()*)
- **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': [
    {
      'order_price': {'base': '8.000 STEEM', 'quote': '9.618 SBD'},
      'real_price': '1.202250000000000004',
      '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'},
  {'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 recieved 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("steemu", "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 sender's 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

They take the following form:

```
{
  "from": "GPH5mgup8evDqMnT86L7scVebRYDC2fwAWmygPEUL43LjstQegYCC",
  "to": "GPH5Ar4j53kFWuEZQ9XhxbAja4YXMPJ2EnUg5QcrdeMFYUNMMNJbe",
  "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)
transaction = block["transactions"][3]
op = transaction["operations"][0]
op_id = op[0]
op_data = op[1]

# block
# transactions
# operation
# operation type
# operation payload
```

(continues on next page)

(continued from previous page)

```
# 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** (*fmt*) – 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.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': ['wss://steemd.pevo.science', 'wss://rpc.buildteam.io', 'wss://steemd.privex.io']}

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 = [{'score': 100, 'version': '0.19.4', 'owner': 'steemit', 'type': 'appba...}
Default configuration

nodes = ['wss://steemd.pevo.science', 'wss://rpc.buildteam.io', 'wss://steemd.privex.io']

class *beem.storage.DataDir*

Bases: *object*

This class ensures that the user's data is stored in its OS preprotected 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={}, 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 = "5KQwrPbwdL6PhXujxW37FSSQZlJiwsST4cqQzDeyXtP79zkvFD3"
stm = Steem(nobroadcast=True, keys={'active': wif})
tx = TransactionBuilder(stem_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 ()

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.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: /.

`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_permalink` (*permalink*)

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("5xxxxxxxxxxxxxxxxxxxxxx")
```

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(*enc_token*)

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

removeTokenFromPublicKey (*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*=*'1.000 STEEM'*, *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, 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 >
```

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.graphenepc.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 instanciating 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.

graphennewsrpc.

```
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.

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*)
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': 19}
```

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', 'withdraw']
```

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', 'withdraw']
```

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

```
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.ecdasig

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(instance)`: 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':

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`, 60
- `beem.aes`, 76
- `beem.amount`, 78
- `beem.asciichart`, 76
- `beem.asset`, 79
- `beem.block`, 90
- `beem.blockchain`, 92
- `beem.blockchainobject`, 97
- `beem.comment`, 98
- `beem.discussions`, 102
- `beem.exceptions`, 107
- `beem.imageuploader`, 109
- `beem.instance`, 110
- `beem.market`, 110
- `beem.memo`, 116
- `beem.message`, 118
- `beem.nodelist`, 88
- `beem.notify`, 118
- `beem.price`, 119
- `beem.steem`, 79
- `beem.steemconnect`, 89
- `beem.storage`, 121
- `beem.transactionbuilder`, 124
- `beem.utils`, 126
- `beem.vote`, 127
- `beem.wallet`, 128
- `beem.witness`, 132
- `beemapi.exceptions`, 135
- `beemapi.graphenerpc`, 138
- `beemapi.node`, 138
- `beemapi.steemnoderpc`, 134
- `beembase.memo`, 140
- `beembase.objects`, 142
- `beembase.objecttypes`, 143
- `beembase.operationids`, 143
- `beembase.signedtransactions`, 143
- `beembase.transactions`, 143
- `beemgraphenebase.account`, 144
- `beemgraphenebase.base58`, 146
- `beemgraphenebase.bip38`, 147
- `beemgraphenebase.objects`, 148
- `beemgraphenebase.objecttypes`, 148
- `beemgraphenebase.operationids`, 148
- `beemgraphenebase.signedtransactions`, 148

Symbols

- account_creation_fee <account_creation_fee>
 - beemply-witnesscreate command line option, 42
 - beemply-witnessupdate command line option, 44
- ascii
 - beemply-orderbook command line option, 31
 - beemply-pricehistory command line option, 34
 - beemply-tradehistory command line option, 38
- auto_vest
 - beemply-powerdownroute command line option, 34
- chart
 - beemply-orderbook command line option, 31
- confirm
 - beemply-delkey command line option, 24
 - beemply-deltoken command line option, 25
- direction <direction>
 - beemply-votes command line option, 41
- fee <fee>
 - beemply-newaccount command line option, 30
- file <file>
 - beemply-broadcast command line option, 21
 - beemply-sign command line option, 37
- hours <hours>
 - beemply-tradehistory command line option, 38
- import-brain-key
 - beemply-keygen command line option, 28
- key <key>
 - beemply-updatememokey command line option, 39
- limit <limit>
 - beemply-witnesses command line option, 43
- maximum_block_size <maximum_block_size>
 - beemply-witnesscreate command line option, 42
 - beemply-witnessupdate command line option, 44
- orderid <orderid>
 - beemply-buy command line option, 21
 - beemply-sell command line option, 36
- payout <payout>
 - beemply-curation command line option, 23
- percentage <percentage>
 - beemply-powerdownroute command line option, 34
- permission <permission>
 - beemply-allow command line option, 19
 - beemply-disallow command line option, 25
- raw
 - beemply-pingnode command line option, 33
- remove
 - beemply-pingnode command line option, 33
- results
 - beemply-nextnode command line option, 30
- reward_sbd <reward_sbd>
 - beemply-claimreward command line option, 22
- reward_steem <reward_steem>
 - beemply-claimreward command line option, 22
- reward_vests <reward_vests>
 - beemply-claimreward command line option, 22
- roles <roles>
 - beemply-importaccount command line option, 28
- sbd_interest_rate <sbd_interest_rate>
 - beemply-witnesscreate command line option, 42
 - beemply-witnessupdate command line option, 44
- sequence <sequence>
 - beemply-keygen command line option, 28
- show-date
 - beemply-orderbook command line option, 31
- signing_key <signing_key>
 - beemply-witnessupdate command line option, 44
- sort
 - beemply-pingnode command line option, 33
- support-peg
 - beemply-witnessfeed command line option, 43
- test-unlock
 - beemply-walletinfo command line option, 41
- threading
 - beemply-pingnode command line option, 33
- threshold <threshold>
 - beemply-allow command line option, 20
 - beemply-disallow command line option, 25
- to <to>
 - beemply-powerup command line option, 34

- unsafe-import-key <unsafe_import_key>
 - beempy-addkey command line option, 19
 - beempy-parsewif command line option, 31
- unsafe-import-token <unsafe_import_token>
 - beempy-addtoken command line option, 19
- url
 - beempy-currentnode command line option, 24
- url <url>
 - beempy-witnesscreate command line option, 42
 - beempy-witnessupdate command line option, 44
- version
 - beempy command line option, 18
 - beempy-currentnode command line option, 24
- weight <weight>
 - beempy-allow command line option, 19
- what <what>
 - beempy-follow command line option, 27
 - beempy-mute command line option, 29
- wipe
 - beempy-createwallet command line option, 23
- witness <witness>
 - beempy-witnessupdate command line option, 44
- a, -account <account>
 - beempy-allow command line option, 19
 - beempy-approvewitness command line option, 20
 - beempy-buy command line option, 21
 - beempy-cancel command line option, 21
 - beempy-convert command line option, 22
 - beempy-curation command line option, 23
 - beempy-delpoint command line option, 25
 - beempy-disallow command line option, 25
 - beempy-disapprovewitness command line option, 26
 - beempy-downvote command line option, 26
 - beempy-follow command line option, 27
 - beempy-mute command line option, 29
 - beempy-newaccount command line option, 30
 - beempy-powerdown command line option, 33
 - beempy-powerdownroute command line option, 34
 - beempy-powerup command line option, 34
 - beempy-resteem command line option, 35
 - beempy-sell command line option, 36
 - beempy-setprofile command line option, 37
 - beempy-transfer command line option, 38
 - beempy-unfollow command line option, 39
 - beempy-updatememokey command line option, 39
 - beempy-upvote command line option, 40
- a, -author
 - beempy-pending command line option, 32
 - beempy-rewards command line option, 35
- a, -only-appbase
 - beempy-updatenodes command line option, 39
- b, -base <base>
 - beempy-witnessfeed command line option, 43
- c, -comment
 - beempy-pending command line option, 32
 - beempy-rewards command line option, 35
- d, -days <days>
 - beempy-curation command line option, 24
 - beempy-pending command line option, 32
 - beempy-rewards command line option, 35
 - beempy-tradehistory command line option, 38
 - beempy-votes command line option, 41
- d, -no-broadcast
 - beempy command line option, 18
- e, -expires <expires>
 - beempy command line option, 18
- e, -export <export>
 - beempy-curation command line option, 23
 - beempy-votes command line option, 41
- e, -permlink
 - beempy-pending command line option, 32
 - beempy-rewards command line option, 35
- h, -height <height>
 - beempy-orderbook command line option, 31
 - beempy-pricehistory command line option, 34
 - beempy-tradehistory command line option, 38
- h, -only-https
 - beempy-updatenodes command line option, 39
- i, -incoming
 - beempy-votes command line option, 41
- i, -sbd-to-steem
 - beempy-ticker command line option, 37
 - beempy-tradehistory command line option, 38
- l, -create-link
 - beempy command line option, 18
- l, -length <length>
 - beempy-curation command line option, 24
 - beempy-pending command line option, 32
 - beempy-rewards command line option, 35
- l, -limit <limit>
 - beempy-orderbook command line option, 31
 - beempy-tradehistory command line option, 38
- m, -limit <limit>
 - beempy-curation command line option, 23
- n, -node <node>
 - beempy command line option, 18
- n, -only-non-appbase
 - beempy-updatenodes command line option, 40
- o, -offline
 - beempy command line option, 18
- o, -outgoing
 - beempy-votes command line option, 41
- p, -no-wallet
 - beempy command line option, 18
- p, -pair <pair>
 - beempy-setprofile command line option, 37
- p, -permlink
 - beempy-curation command line option, 24

-p, --post
 beem-pending command line option, 32
 beem-rewards command line option, 35

-q, --quote <quote>
 beem-witnessfeed command line option, 43

-s, --only-sum
 beem-pending command line option, 32
 beem-rewards command line option, 35

-s, --short
 beem-curation command line option, 23

-s, --show
 beem-updatenodes command line option, 39

-s, --steemconnect
 beem command line option, 18

-t, --test
 beem-updatenodes command line option, 39

-t, --title
 beem-curation command line option, 24
 beem-pending command line option, 32
 beem-rewards command line option, 35

-t, --trx <trx>
 beem-verify command line option, 40

-u, --use-api
 beem-verify command line option, 40

-v, --curation
 beem-pending command line option, 32
 beem-rewards command line option, 35

-v, --min-vote <min_vote>
 beem-curation command line option, 23

-v, --verbose <verbose>
 beem command line option, 18

-w, --max-vote <max_vote>
 beem-curation command line option, 23

-w, --only-wss
 beem-updatenodes command line option, 39

-w, --weight <weight>
 beem-downvote command line option, 26
 beem-upvote command line option, 40

-w, --width <width>
 beem-orderbook command line option, 31
 beem-pricehistory command line option, 34
 beem-tradehistory command line option, 38

-x, --min-performance <min_performance>
 beem-curation command line option, 23

-x, --unsigned
 beem command line option, 18

-y, --max-performance <max_performance>
 beem-curation command line option, 23

__SteemWebsocket__set_subscriptions()
 (beemapi.websocket.SteemWebsocket
 method), 136

__events__ (beemapi.websocket.SteemWebsocket attribute), 136

__getattr__() (beemapi.websocket.SteemWebsocket
 method), 137

__init__() (beemapi.websocket.SteemWebsocket
 method), 137

__module__ (beemapi.websocket.SteemWebsocket attribute), 137

_ping() (beemapi.websocket.SteemWebsocket method), 137

A

abort() (beem.blockchain.Pool method), 96

ACCOUNT

beem-balance command line option, 20
 beem-claimreward command line option, 22
 beem-follower command line option, 27
 beem-following command line option, 27
 beem-importaccount command line option, 28
 beem-interest command line option, 28
 beem-muter command line option, 29
 beem-muting command line option, 30
 beem-openorders command line option, 31
 beem-permissions command line option, 32
 beem-power command line option, 33
 beem-votes command line option, 41
 beem-witnesses command line option, 43

account (beem.witness.Witness attribute), 133

Account (class in beem.account), 60

AccountDoesNotExistException, 107

AccountExistsException, 107

ACCOUNTNAME

beem-newaccount command line option, 30

accountopenorders() (beem.market.Market method), 111

ACCOUNTS

beem-pending command line option, 32

beem-rewards command line option, 36

Accounts (class in beem.account), 75

AccountsObject (class in beem.account), 76

AccountVotes (class in beem.vote), 127

ActiveVotes (class in beem.vote), 127

adapt_on_series() (beem.asciichart.AsciiChart method), 76

add() (beem.storage.Key method), 122

add() (beem.storage.Token method), 123

add_axis() (beem.asciichart.AsciiChart method), 77

add_curve() (beem.asciichart.AsciiChart method), 77

addPrivateKey() (beem.wallet.Wallet method), 129

Address (class in beemgraphenebase.account), 144

addSigningInformation()
 (beem.transactionbuilder.TransactionBuilder
 method), 125

addToken() (beem.wallet.Wallet method), 129

addTzInfo() (in module beem.utils), 126

AESCipher (class in beem.aes), 76

alive() (beem.blockchain.Pool method), 96

`allow()` (beem.account.Account method), 60

AMOUNT

- beem-py-buy command line option, 21
- beem-py-convert command line option, 23
- beem-py-powerdown command line option, 33
- beem-py-powerup command line option, 34
- beem-py-sell command line option, 36
- beem-py-transfer command line option, 38

`amount` (beem.amount.Amount attribute), 78

`Amount` (class in beem.amount), 78

`Amount` (class in beembase.objects), 142

`ApiNotSupported`, 135

`appauthor` (beem.storage.DataDir attribute), 122

`appendMissingSignatures()`

(beem.transactionbuilder.TransactionBuilder method), 125

`appendOps()` (beem.transactionbuilder.TransactionBuilder method), 125

`appendSigner()` (beem.transactionbuilder.TransactionBuilder method), 125

`appendWif()` (beem.transactionbuilder.TransactionBuilder method), 125

`appname` (beem.storage.DataDir attribute), 122

`approveWitness()` (beem.account.Account method), 61

`as_base()` (beem.price.Price method), 120

`as_quote()` (beem.price.Price method), 120

`AsciiChart` (class in beem.asciichart), 76

ASSET

- beem-py-buy command line option, 21
- beem-py-sell command line option, 36
- beem-py-transfer command line option, 38

`asset` (beem.amount.Amount attribute), 79

`asset` (beem.asset.Asset attribute), 79

`Asset` (class in beem.asset), 79

`AssetDoesNotExistsException`, 107

`assets_from_string()` (in module beem.utils), 126

`author` (beem.comment.Comment attribute), 98

AUTHORPERM

- beem-py-curation command line option, 24

`authorperm` (beem.comment.Comment attribute), 98

`authorperm` (beem.vote.Vote attribute), 128

`available_balances` (beem.account.Account attribute), 61

`awaitTxConfirmation()` (beem.blockchain.Blockchain method), 93

B

`b58decode()` (in module beemgraphenebase.base58), 147

`b58encode()` (in module beemgraphenebase.base58), 147

`balances` (beem.account.Account attribute), 61

`Base58` (class in beemgraphenebase.base58), 146

`base58CheckDecode()` (in module beemgraphenebase.base58), 147

`base58CheckEncode()` (in module beemgraphenebase.base58), 147

`base58decode()` (in module beemgraphenebase.base58), 147

`base58encode()` (in module beemgraphenebase.base58), 147

`BatchedCallsNotSupported`, 108

`beem.account` (module), 60

`beem.aes` (module), 76

`beem.amount` (module), 78

`beem.asciichart` (module), 76

`beem.asset` (module), 79

`beem.block` (module), 90

`beem.blockchain` (module), 92

`beem.blockchainobject` (module), 97

`beem.comment` (module), 98

`beem.discussions` (module), 102

`beem.exceptions` (module), 107

`beem.imageuploader` (module), 109

`beem.instance` (module), 110

`beem.market` (module), 110

`beem.memo` (module), 116

`beem.message` (module), 118

`beem.nodelist` (module), 88

`beem.notify` (module), 118

`beem.price` (module), 119

`beem.steem` (module), 79

`beem.steemconnect` (module), 89

`beem.storage` (module), 121

`beem.transactionbuilder` (module), 124

`beem.utils` (module), 126

`beem.vote` (module), 127

`beem.wallet` (module), 128

`beem.witness` (module), 132

`beemapi.exceptions` (module), 135

`beemapi.graphenerpc` (module), 138

`beemapi.node` (module), 138

`beemapi.steemnodeRPC` (module), 134

`beembase.memo` (module), 140

`beembase.objects` (module), 142

`beembase.objecttypes` (module), 143

`beembase.operationids` (module), 143

`beembase.signedtransactions` (module), 143

`beembase.transactions` (module), 143

`beemgraphenebase.account` (module), 144

`beemgraphenebase.base58` (module), 146

`beemgraphenebase.bip38` (module), 147

`beemgraphenebase.objects` (module), 148

`beemgraphenebase.objecttypes` (module), 148

`beemgraphenebase.operationids` (module), 148

`beemgraphenebase.signedtransactions` (module), 148

`beem-py` command line option

–version, 18

–d, –no-broadcast, 18

–e, –expires <expires>, 18

–l, –create-link, 18

- n, --node <node>, 18
- o, --offline, 18
- p, --no-wallet, 18
- s, --steemconnect, 18
- v, --verbose <verbose>, 18
- x, --unsigned, 18
- beem-`addkey` command line option
 - unsafe-import-key <unsafe_import_key>, 19
- beem-`addtoken` command line option
 - unsafe-import-token <unsafe_import_token>, 19
 - NAME, 19
- beem-`allow` command line option
 - permission <permission>, 19
 - threshold <threshold>, 20
 - weight <weight>, 19
 - a, --account <account>, 19
 - FOREIGN_ACCOUNT, 20
- beem-`approvewitness` command line option
 - a, --account <account>, 20
 - WITNESS, 20
- beem-`balance` command line option
 - ACCOUNT, 20
- beem-`broadcast` command line option
 - file <file>, 21
- beem-`buy` command line option
 - orderid <orderid>, 21
 - a, --account <account>, 21
 - AMOUNT, 21
 - ASSET, 21
 - PRICE, 21
- beem-`cancel` command line option
 - a, --account <account>, 21
 - ORDERID, 21
- beem-`claimreward` command line option
 - reward_sbd <reward_sbd>, 22
 - reward_steem <reward_steem>, 22
 - reward_vests <reward_vests>, 22
 - ACCOUNT, 22
- beem-`convert` command line option
 - a, --account <account>, 22
 - AMOUNT, 23
- beem-`createwallet` command line option
 - wipe, 23
- beem-`curation` command line option
 - payout <payout>, 23
 - a, --account <account>, 23
 - d, --days <days>, 24
 - e, --export <export>, 23
 - l, --length <length>, 24
 - m, --limit <limit>, 23
 - p, --permlink, 24
 - s, --short, 23
 - t, --title, 24
 - v, --min-vote <min_vote>, 23
 - w, --max-vote <max_vote>, 23
 - x, --min-performance <min_performance>, 23
 - y, --max-performance <max_performance>, 23
 - AUTHORPERM, 24
- beem-`currentnode` command line option
 - url, 24
 - version, 24
- beem-`delkey` command line option
 - confirm, 24
 - PUB, 24
- beem-`delpoint` command line option
 - a, --account <account>, 25
 - VARIABLE, 25
- beem-`deltoken` command line option
 - confirm, 25
 - NAME, 25
- beem-`disallow` command line option
 - permission <permission>, 25
 - threshold <threshold>, 25
 - a, --account <account>, 25
 - FOREIGN_ACCOUNT, 26
- beem-`disapprovewitness` command line option
 - a, --account <account>, 26
 - WITNESS, 26
- beem-`downvote` command line option
 - a, --account <account>, 26
 - w, --weight <weight>, 26
 - POST, 26
 - VOTE_WEIGHT, 26
- beem-`follow` command line option
 - what <what>, 27
 - a, --account <account>, 27
 - FOLLOW, 27
- beem-`follower` command line option
 - ACCOUNT, 27
- beem-`following` command line option
 - ACCOUNT, 27
- beem-`importaccount` command line option
 - roles <roles>, 28
 - ACCOUNT, 28
- beem-`info` command line option
 - OBJECTS, 28
- beem-`interest` command line option
 - ACCOUNT, 28
- beem-`keygen` command line option
 - import-brain-key, 28
 - sequence <sequence>, 28
- beem-`mute` command line option
 - what <what>, 29
 - a, --account <account>, 29
 - MUTE, 29
- beem-`muter` command line option
 - ACCOUNT, 29
- beem-`muting` command line option

- ACCOUNT, 30
- beem-py-newaccount command line option
 - fee <fee>, 30
 - a, -account <account>, 30
 - ACCOUNTNAME, 30
- beem-py-nextnode command line option
 - results, 30
- beem-py-openorders command line option
 - ACCOUNT, 31
- beem-py-orderbook command line option
 - ascii, 31
 - chart, 31
 - show-date, 31
 - h, -height <height>, 31
 - l, -limit <limit>, 31
 - w, -width <width>, 31
- beem-py-parsewif command line option
 - unsafe-import-key <unsafe_import_key>, 31
- beem-py-pending command line option
 - a, -author, 32
 - c, -comment, 32
 - d, -days <days>, 32
 - e, -permlink, 32
 - l, -length <length>, 32
 - p, -post, 32
 - s, -only-sum, 32
 - t, -title, 32
 - v, -curation, 32
 - ACCOUNTS, 32
- beem-py-permissions command line option
 - ACCOUNT, 32
- beem-py-pingnode command line option
 - raw, 33
 - remove, 33
 - sort, 33
 - threading, 33
- beem-py-power command line option
 - ACCOUNT, 33
- beem-py-powerdown command line option
 - a, -account <account>, 33
 - AMOUNT, 33
- beem-py-powerdownroute command line option
 - auto_vest, 34
 - percentage <percentage>, 34
 - a, -account <account>, 34
 - TO, 34
- beem-py-powerup command line option
 - to <to>, 34
 - a, -account <account>, 34
 - AMOUNT, 34
- beem-py-pricehistory command line option
 - ascii, 34
 - h, -height <height>, 34
 - w, -width <width>, 34
- beem-py-resteem command line option
 - a, -account <account>, 35
 - IDENTIFIER, 35
- beem-py-rewards command line option
 - a, -author, 35
 - c, -comment, 35
 - d, -days <days>, 35
 - e, -permlink, 35
 - l, -length <length>, 35
 - p, -post, 35
 - s, -only-sum, 35
 - t, -title, 35
 - v, -curation, 35
 - ACCOUNTS, 36
- beem-py-sell command line option
 - orderid <orderid>, 36
 - a, -account <account>, 36
 - AMOUNT, 36
 - ASSET, 36
 - PRICE, 36
- beem-py-set command line option
 - KEY, 36
 - VALUE, 36
- beem-py-setprofile command line option
 - a, -account <account>, 37
 - p, -pair <pair>, 37
 - VALUE, 37
 - VARIABLE, 37
- beem-py-sign command line option
 - file <file>, 37
- beem-py-ticker command line option
 - i, -sbd-to-steem, 37
- beem-py-tradehistory command line option
 - ascii, 38
 - hours <hours>, 38
 - d, -days <days>, 38
 - h, -height <height>, 38
 - i, -sbd-to-steem, 38
 - l, -limit <limit>, 38
 - w, -width <width>, 38
- beem-py-transfer command line option
 - a, -account <account>, 38
 - AMOUNT, 38
 - ASSET, 38
 - MEMO, 38
 - TO, 38
- beem-py-unfollow command line option
 - a, -account <account>, 39
 - UNFOLLOW, 39
- beem-py-updatememokey command line option
 - key <key>, 39
 - a, -account <account>, 39
- beem-py-updatenodes command line option
 - a, -only-appbase, 39

- h, --only-https, 39
 - n, --only-non-appbase, 40
 - s, --show, 39
 - t, --test, 39
 - w, --only-wss, 39
 - beem-py-upvote command line option
 - a, --account <account>, 40
 - w, --weight <weight>, 40
 - POST, 40
 - VOTE_WEIGHT, 40
 - beem-py-verify command line option
 - t, --trx <trx>, 40
 - u, --use-api, 40
 - BLOCKNUMBER, 40
 - beem-py-votes command line option
 - direction <direction>, 41
 - d, --days <days>, 41
 - e, --export <export>, 41
 - i, --incoming, 41
 - o, --outgoing, 41
 - ACCOUNT, 41
 - beem-py-walletinfo command line option
 - test-unlock, 41
 - beem-py-witness command line option
 - WITNESS, 41
 - beem-py-witnesscreate command line option
 - account_creation_fee <account_creation_fee>, 42
 - maximum_block_size <maximum_block_size>, 42
 - sbd_interest_rate <sbd_interest_rate>, 42
 - url <url>, 42
 - PUB_SIGNING_KEY, 42
 - WITNESS, 42
 - beem-py-witnessdisable command line option
 - WITNESS, 42
 - beem-py-witnessenable command line option
 - SIGNING_KEY, 42
 - WITNESS, 42
 - beem-py-witnesses command line option
 - limit <limit>, 43
 - ACCOUNT, 43
 - beem-py-witnessfeed command line option
 - support-peg, 43
 - b, --base <base>, 43
 - q, --quote <quote>, 43
 - WITNESS, 43
 - beem-py-witnessupdate command line option
 - account_creation_fee <account_creation_fee>, 44
 - maximum_block_size <maximum_block_size>, 44
 - sbd_interest_rate <sbd_interest_rate>, 44
 - signing_key <signing_key>, 44
 - url <url>, 44
 - witness <witness>, 44
 - Beneficiaries (class in beembase.objects), 142
 - Beneficiary (class in beembase.objects), 142
 - Block (class in beem.block), 90
 - block_num (beem.block.Block attribute), 91
 - block_num (beem.block.BlockHeader attribute), 92
 - block_time() (beem.blockchain.Blockchain method), 93
 - block_timestamp() (beem.blockchain.Blockchain method), 93
 - Blockchain (class in beem.blockchain), 92
 - BlockchainObject (class in beem.blockchainobject), 97
 - BlockDoesNotExistException, 108
 - BlockHeader (class in beem.block), 91
 - BLOCKNUMBER
 - beem-py-verify command line option, 40
 - blocks() (beem.blockchain.Blockchain method), 93
 - BlockWaitTimeExceeded, 108
 - body (beem.comment.Comment attribute), 98
 - BrainKey (class in beemgraphenebase.account), 144
 - broadcast() (beem.steem.Steem method), 81
 - broadcast() (beem.steemconnect.SteemConnect method), 89
 - broadcast() (beem.transactionbuilder.TransactionBuilder method), 125
 - btc_usd_ticker() (beem.market.Market static method), 111
 - buy() (beem.market.Market method), 111
- ## C
- cache() (beem.blockchainobject.BlockchainObject method), 97
 - CallRetriesReached, 135
 - cancel() (beem.market.Market method), 112
 - cancel_subscriptions() (beemapi.websocket.SteemWebsocket method), 137
 - cancel_transfer_from_savings() (beem.account.Account method), 61
 - category (beem.comment.Comment attribute), 98
 - chain_params (beem.steem.Steem attribute), 81
 - changePassphrase() (beem.wallet.Wallet method), 129
 - changePassword() (beem.storage.MasterPassword method), 123
 - checkBackup() (beem.storage.Configuration method), 121
 - child() (beemgraphenebase.account.PrivateKey method), 145
 - claim_reward_balance() (beem.account.Account method), 61
 - clean_data() (beem.storage.DataDir method), 122
 - clear() (beem.steem.Steem method), 81
 - clear() (beem.transactionbuilder.TransactionBuilder method), 125
 - clear_cache() (beem.blockchainobject.BlockchainObject static method), 97
 - clear_cache() (in module beem.instance), 110
 - clear_cache_from_expired_items() (beem.blockchainobject.BlockchainObject

- method), 97
- clear_data() (beem.asciichart.AsciiChart method), 77
- clear_expired_items() (beem.blockchainobject.ObjectCache method), 98
- clear_local_keys() (beem.wallet.Wallet method), 129
- clear_local_token() (beem.wallet.Wallet method), 129
- clearWifs() (beem.transactionbuilder.TransactionBuilder method), 125
- close() (beem.notify.Notify method), 119
- close() (beemapi.websocket.SteemWebsocket method), 137
- Comment (class in beem.comment), 98
- Comment_discussions_by_payout (class in beem.discussions), 102
- comment_options() (beem.steem.Steem method), 81
- CommentOptionExtensions (class in beembase.objects), 142
- compressed() (beemgraphenebase.account.PublicKey method), 146
- compressedpubkey() (beemgraphenebase.account.PrivateKey method), 145
- config (beem.instance.SharedInstance attribute), 110
- config_defaults (beem.storage.Configuration attribute), 121
- config_key (beem.storage.MasterPassword attribute), 123
- configStorage (beem.wallet.Wallet attribute), 129
- Configuration (class in beem.storage), 121
- connect() (beem.steem.Steem method), 81
- construct_authorperm() (in module beem.utils), 126
- construct_authorpermvoter() (in module beem.utils), 126
- constructTx() (beem.transactionbuilder.TransactionBuilder method), 125
- ContentDoesNotExistsException, 108
- convert() (beem.account.Account method), 61
- copy() (beem.amount.Amount method), 79
- copy() (beem.price.Price method), 121
- create() (beem.wallet.Wallet method), 129
- create_account() (beem.steem.Steem method), 81
- create_hot_sign_url() (beem.steemconnect.SteemConnect method), 90
- create_table() (beem.storage.Configuration method), 121
- create_table() (beem.storage.Key method), 122
- create_table() (beem.storage.Token method), 124
- create_ws_instance() (in module beemapi.graphenerrpc), 140
- created() (beem.wallet.Wallet method), 129
- curation_penalty_compensation_SBD() (beem.comment.Comment method), 98
- curation_stats() (beem.account.Account method), 62
- custom_json() (beem.steem.Steem method), 82
- DataDir (class in beem.storage), 121
- decode_memo() (in module beembase.memo), 140
- decode_memo_bts() (in module beembase.memo), 140
- decodeRPCErrorMsg() (in module beemapi.exceptions), 136
- decrypt() (beem.aes.AESCipher method), 76
- decrypt() (beem.memo.Memo method), 118
- decrypt() (in module beemgraphenebase.bip38), 147
- decrypt_token() (beem.wallet.Wallet method), 130
- decrypt_wif() (beem.wallet.Wallet method), 130
- decrypted_master (beem.storage.MasterPassword attribute), 123
- decryptEncryptedMaster() (beem.storage.MasterPassword method), 123
- default_handler() (in module beem.blockchain), 97
- delegate_vesting_shares() (beem.account.Account method), 62
- delete() (beem.comment.Comment method), 98
- delete() (beem.storage.Configuration method), 121
- delete() (beem.storage.Key method), 122
- delete() (beem.storage.Token method), 124
- derive256address_with_version() (beemgraphenebase.account.Address method), 144
- derive_from_seed() (beemgraphenebase.account.PrivateKey method), 145
- derive_permlink() (in module beem.utils), 126
- derive_private_key() (beemgraphenebase.account.PrivateKey method), 145
- deriveChecksum() (beem.storage.MasterPassword method), 123
- deriveChecksum() (beem.wallet.Wallet method), 130
- deriveDigest() (beemgraphenebase.signedtransactions.Signed_Transaction method), 148
- derivesha256address() (beemgraphenebase.account.Address method), 144
- derivesha512address() (beemgraphenebase.account.Address method), 144
- derSigToHexSig() (beemgraphenebase.signedtransactions.Signed_Transaction method), 148
- disallow() (beem.account.Account method), 62
- disapprovewitness() (beem.account.Account method), 62
- Discussions (class in beem.discussions), 102
- Discussions_by_active (class in beem.discussions), 103
- Discussions_by_author_before_date (class in beem.discussions), 103
- Discussions_by_blog (class in beem.discussions), 103
- Discussions_by_cashout (class in beem.discussions), 103

D

Discussions_by_children (class in beem.discussions), 104
 Discussions_by_comments (class in beem.discussions), 104
 Discussions_by_created (class in beem.discussions), 104
 Discussions_by_feed (class in beem.discussions), 105
 Discussions_by_hot (class in beem.discussions), 105
 Discussions_by_promoted (class in beem.discussions), 105
 Discussions_by_trending (class in beem.discussions), 105
 Discussions_by_votes (class in beem.discussions), 106
 done() (beem.blockchain.Pool method), 96
 doublesha256() (in module beemgraphenebase.base58), 147
 downvote() (beem.comment.Comment method), 98

E

edit() (beem.comment.Comment method), 99
 encode_memo() (in module beembase.memo), 141
 encode_memo_bts() (in module beembase.memo), 141
 encrypt() (beem.aes.AESCipher method), 76
 encrypt() (beem.memo.Memo method), 118
 encrypt() (in module beemgraphenebase.bip38), 147
 encrypt_token() (beem.wallet.Wallet method), 130
 encrypt_wif() (beem.wallet.Wallet method), 130
 enqueue() (beem.blockchain.Pool method), 96
 ensure_full() (beem.account.Account method), 62
 error_cnt (beemapi.graphenerpc.GrapheneRPC attribute), 139
 error_cnt (beemapi.node.Nodes attribute), 138
 error_cnt_call (beemapi.graphenerpc.GrapheneRPC attribute), 139
 error_cnt_call (beemapi.node.Nodes attribute), 138
 estimate_curation_SBD() (beem.comment.Comment method), 99
 estimate_virtual_op_num() (beem.account.Account method), 62
 ExchangeRate (class in beembase.objects), 142
 exists_table() (beem.storage.Configuration method), 121
 exists_table() (beem.storage.Key method), 122
 exists_table() (beem.storage.Token method), 124
 export_working_nodes() (beemapi.node.Nodes method), 138
 Extension (class in beembase.objects), 142

F

feed_publish() (beem.witness.Witness method), 133
 FilledOrder (class in beem.price), 119
 finalizeOp() (beem.steem.Steem method), 83
 findall_patch_hunks() (in module beem.utils), 126
 FOLLOW
 beempy-follow command line option, 27
 follow() (beem.account.Account method), 63
 FOREIGN_ACCOUNT

beempy-allow command line option, 20
 beempy-disallow command line option, 26
 formatTime() (in module beem.utils), 127
 formatTimedelta() (in module beem.utils), 127
 formatTimeFromNow() (in module beem.utils), 127
 formatTimeString() (in module beem.utils), 127

G

get() (beem.blockchainobject.ObjectCache method), 98
 get() (beem.storage.Configuration method), 121
 get_access_token() (beem.steemconnect.SteemConnect method), 90
 get_account() (beemapi.steemnodepc.SteemNodeRPC method), 134
 get_account_bandwidth() (beem.account.Account method), 63
 get_account_count() (beem.blockchain.Blockchain method), 93
 get_account_history() (beem.account.Account method), 63
 get_account_reputations() (beem.blockchain.Blockchain method), 93
 get_account_votes() (beem.account.Account method), 64
 get_all_accounts() (beem.blockchain.Blockchain method), 94
 get_author_rewards() (beem.comment.Comment method), 99
 get_balance() (beem.account.Account method), 64
 get_balances() (beem.account.Account method), 64
 get_bandwidth() (beem.account.Account method), 64
 get_beneficiaries_pct() (beem.comment.Comment method), 99
 get_blind_private() (beem-graphenebase.account.BrainKey method), 144
 get_block_interval() (beem.steem.Steem method), 83
 get_blockchain_version() (beem.steem.Steem method), 83
 get_blog() (beem.account.Account method), 65
 get_blog_authors() (beem.account.Account method), 65
 get_blog_entries() (beem.account.Account method), 65
 get_brainkey() (beemgraphenebase.account.BrainKey method), 144
 get_cache_auto_clean() (beem.blockchainobject.BlockchainObject method), 97
 get_cache_expiration() (beem.blockchainobject.BlockchainObject method), 97
 get_chain_properties() (beem.steem.Steem method), 83
 get_config() (beem.steem.Steem method), 83
 get_conversion_requests() (beem.account.Account method), 66
 get_creator() (beem.account.Account method), 66
 get_curation_penalty() (beem.comment.Comment method), 99

`get_curation_reward()` (beem.account.Account method), 66

`get_curation_rewards()` (beem.comment.Comment method), 99

`get_current_block()` (beem.blockchain.Blockchain method), 94

`get_current_block_num()` (beem.blockchain.Blockchain method), 94

`get_current_median_history()` (beem.steem.Steem method), 83

`get_default_nodes()` (beem.steem.Steem method), 84

`get_discussions()` (beem.discussions.Discussions method), 102

`get_dynamic_global_properties()` (beem.steem.Steem method), 84

`get_escrow()` (beem.account.Account method), 66

`get_estimated_block_num()` (beem.blockchain.Blockchain method), 94

`get_expiring_vesting_delegations()` (beem.account.Account method), 66

`get_feed()` (beem.account.Account method), 66

`get_feed_entries()` (beem.account.Account method), 67

`get_feed_history()` (beem.steem.Steem method), 84

`get_follow_count()` (beem.account.Account method), 67

`get_followers()` (beem.account.Account method), 67

`get_following()` (beem.account.Account method), 67

`get_hardfork_properties()` (beem.steem.Steem method), 84

`get_list()` (beem.vote.VotesObject method), 128

`get_login_url()` (beem.steemconnect.SteemConnect method), 90

`get_median_price()` (beem.steem.Steem method), 84

`get_muters()` (beem.account.Account method), 67

`get_mutings()` (beem.account.Account method), 67

`get_network()` (beem.steem.Steem method), 84

`get_network()` (beemapi.graphenerpc.GrapheneRPC method), 139

`get_nodes()` (beem.nodelist.NodeList method), 88

`get_owner_history()` (beem.account.Account method), 67

`get_parent()` (beem.transactionbuilder.TransactionBuilder method), 125

`get_potential_signatures()` (beem.transactionbuilder.TransactionBuilder method), 125

`get_private()` (beemgraphenebase.account.BrainKey method), 144

`get_private()` (beemgraphenebase.account.PasswordKey method), 145

`get_private_key()` (beemgraphenebase.account.BrainKey method), 144

`get_private_key()` (beemgraphenebase.account.PasswordKey method), 145

`get_public()` (beemgraphenebase.account.BrainKey method), 144

`get_public()` (beemgraphenebase.account.PasswordKey method), 145

`get_public_key()` (beemgraphenebase.account.Address method), 144

`get_public_key()` (beemgraphenebase.account.BrainKey method), 145

`get_public_key()` (beemgraphenebase.account.PasswordKey method), 145

`get_public_key()` (beemgraphenebase.account.PrivateKey method), 145

`get_public_key()` (beemgraphenebase.account.PublicKey method), 146

`get_reblogged_by()` (beem.comment.Comment method), 100

`get_recharge_time()` (beem.account.Account method), 68

`get_recharge_time_str()` (beem.account.Account method), 68

`get_recharge_timedelta()` (beem.account.Account method), 68

`get_recovery_request()` (beem.account.Account method), 68

`get_replies()` (beem.comment.Comment method), 100

`get_reputation()` (beem.account.Account method), 68

`get_request_id()` (beemapi.graphenerpc.GrapheneRPC method), 139

`get_request_id()` (beemapi.websocket.SteemWebsocket method), 137

`get_required_signatures()` (beem.transactionbuilder.TransactionBuilder method), 125

`get_reserve_ratio()` (beem.steem.Steem method), 84

`get_reward_funds()` (beem.steem.Steem method), 84

`get_rewards()` (beem.comment.Comment method), 100

`get_savings_withdrawals()` (beem.account.Account method), 68

`get_sbd_per_rshares()` (beem.steem.Steem method), 84

`get_secret()` (beemgraphenebase.account.PrivateKey method), 146

`get_shared_secret()` (in module beembase.memo), 141

`get_similar_account_names()` (beem.account.Account method), 68

`get_similar_account_names()` (beem.blockchain.Blockchain method), 94

`get_sorted_list()` (beem.vote.VotesObject method), 128

`get_steem_per_mvest()` (beem.steem.Steem method), 84

`get_steem_power()` (beem.account.Account method), 69

`get_string()` (beem.market.Market method), 112

`get_tags_used_by_author()` (beem.account.Account method), 69

`get_testnet()` (beem.nodelist.NodeList method), 88

`get_transaction()` (beem.blockchain.Blockchain method), 95

- [get_transaction_hex\(\)](#) (beem.blockchain.Blockchain method), 95
[get_transaction_hex\(\)](#) (beem.transactionbuilder.TransactionBuilder method), 126
[get_use_appbase\(\)](#) (beemapi.graphenerpc.GrapheneRPC method), 139
[get_vesting_delegations\(\)](#) (beem.account.Account method), 69
[get_vote\(\)](#) (beem.account.Account method), 69
[get_vote_with_curation\(\)](#) (beem.comment.Comment method), 100
[get_votes\(\)](#) (beem.comment.Comment method), 100
[get_votes_sum\(\)](#) (beem.witness.WitnessesObject method), 133
[get_voting_power\(\)](#) (beem.account.Account method), 69
[get_voting_value_SBD\(\)](#) (beem.account.Account method), 69
[get_withdraw_routes\(\)](#) (beem.account.Account method), 69
[get_witness_schedule\(\)](#) (beem.steem.Steem method), 84
[getAccount\(\)](#) (beem.wallet.Wallet method), 130
[getAccountFromPrivateKey\(\)](#) (beem.wallet.Wallet method), 130
[getAccountFromPublicKey\(\)](#) (beem.wallet.Wallet method), 130
[getAccounts\(\)](#) (beem.wallet.Wallet method), 130
[getAccountsFromPublicKey\(\)](#) (beem.wallet.Wallet method), 130
[getActiveKeyForAccount\(\)](#) (beem.wallet.Wallet method), 130
[getActiveKeysForAccount\(\)](#) (beem.wallet.Wallet method), 130
[getAllAccounts\(\)](#) (beem.wallet.Wallet method), 130
[getBlockParams\(\)](#) (in module beembase.transactions), 143
[getcache\(\)](#) (beem.blockchainobject.BlockchainObject method), 97
[getChainParams\(\)](#) (beem-graphenebase.signedtransactions.Signed_Transaction method), 148
[getEncryptedMaster\(\)](#) (beem.storage.MasterPassword method), 123
[getKeyForAccount\(\)](#) (beem.wallet.Wallet method), 130
[getKeysForAccount\(\)](#) (beem.wallet.Wallet method), 131
[getKeyType\(\)](#) (beem.wallet.Wallet method), 130
[getKnownChains\(\)](#) (beem-base.signedtransactions.Signed_Transaction method), 143
[getKnownChains\(\)](#) (beem-graphenebase.signedtransactions.Signed_Transaction method), 148
[getMemoKeyForAccount\(\)](#) (beem.wallet.Wallet method), 131
[getOperationKlass\(\)](#) (beem-base.signedtransactions.Signed_Transaction method), 143
[getOperationKlass\(\)](#) (beem-graphenebase.signedtransactions.Signed_Transaction method), 149
[getOperationNameForId\(\)](#) (beembase.objects.Operation method), 142
[getOperationNameForId\(\)](#) (beem-graphenebase.objects.Operation method), 148
[getOperationNameForId\(\)](#) (in module beem-base.operationids), 143
[getOwnerKeyForAccount\(\)](#) (beem.wallet.Wallet method), 131
[getOwnerKeysForAccount\(\)](#) (beem.wallet.Wallet method), 131
[getPostingKeyForAccount\(\)](#) (beem.wallet.Wallet method), 131
[getPostingKeysForAccount\(\)](#) (beem.wallet.Wallet method), 131
[getPrivateKeyForPublicKey\(\)](#) (beem.storage.Key method), 122
[getPrivateKeyForPublicKey\(\)](#) (beem.wallet.Wallet method), 131
[getPublicKeys\(\)](#) (beem.storage.Key method), 123
[getPublicKeys\(\)](#) (beem.wallet.Wallet method), 131
[getPublicNames\(\)](#) (beem.storage.Token method), 124
[getPublicNames\(\)](#) (beem.wallet.Wallet method), 131
[getSimilarAccountNames\(\)](#) (beem.account.Account method), 63
[getTokenForAccountName\(\)](#) (beem.wallet.Wallet method), 131
[getTokenForPublicName\(\)](#) (beem.storage.Token method), 124
[gphBase58CheckDecode\(\)](#) (in module beem-graphenebase.base58), 147
[gphBase58CheckEncode\(\)](#) (in module beem-graphenebase.base58), 147
[GrapheneObject](#) (class in beemgraphenebase.objects), 148
[GrapheneRPC](#) (class in beemapi.graphenerpc), 138
- ## H
- [has_voted\(\)](#) (beem.account.Account method), 69
[hash_op\(\)](#) (beem.blockchain.Blockchain static method), 95
[headers](#) (beem.steemconnect.SteemConnect attribute), 90
[history\(\)](#) (beem.account.Account method), 70
[history_reverse\(\)](#) (beem.account.Account method), 71
- ## I
- [id](#) (beem.comment.Comment attribute), 100
[id](#) (beemgraphenebase.signedtransactions.Signed_Transaction attribute), 149

IDENTIFIER

beem-py-resteem command line option, 35
idle() (beem.blockchain.Pool method), 97
ImageUploader (class in beem.imageuploader), 109
increase_error_cnt() (beemapi.node.Nodes method), 138
increase_error_cnt_call() (beemapi.node.Nodes method), 138
info() (beem.steem.Steem method), 84
init_aes() (in module beembase.memo), 141
init_aes_bts() (in module beembase.memo), 141
instance (beem.instance.SharedInstance attribute), 110
instance (beemapi.graphenerpc.SessionInstance attribute), 140
InsufficientAuthorityError, 108
interest() (beem.account.Account method), 72
InvalidAssetException, 108
InvalidEndpointUrl, 135
InvalidMemoKeyException, 108
InvalidMessageSignature, 108
InvalidWifiError, 108
invert() (beem.price.Price method), 121
is_active (beem.witness.Witness attribute), 133
is_appbase_ready() (beemapi.graphenerpc.GrapheneRPC method), 139
is_comment() (beem.comment.Comment method), 100
is_connected() (beem.steem.Steem method), 84
is_empty() (beem.transactionbuilder.TransactionBuilder method), 126
is_fully_loaded (beem.account.Account attribute), 72
is_irreversible_mode() (beem.blockchain.Blockchain method), 95
is_main_post() (beem.comment.Comment method), 100
is_pending() (beem.comment.Comment method), 101
isArgsThisClass() (in module beem-graphenebase.objects), 148
iscached() (beem.blockchainobject.BlockchainObject method), 97
items() (beem.blockchainobject.BlockchainObject method), 97
items() (beem.storage.Configuration method), 121

J

join() (beem.blockchain.Pool method), 97
json() (beem.account.Account method), 72
json() (beem.amount.Amount method), 79
json() (beem.block.Block method), 91
json() (beem.block.BlockHeader method), 92
json() (beem.blockchainobject.BlockchainObject method), 97
json() (beem.comment.Comment method), 101
json() (beem.price.FilledOrder method), 119
json() (beem.price.Price method), 121
json() (beem.transactionbuilder.TransactionBuilder method), 126

json() (beem.vote.Vote method), 128
json() (beem.witness.Witness method), 133
json() (beembase.objects.Operation method), 142
json() (beemgraphenebase.objects.GrapheneObject method), 148
json_metadata (beem.account.Account attribute), 72
json_metadata (beem.comment.Comment attribute), 101
json_operations (beem.block.Block attribute), 91
json_transactions (beem.block.Block attribute), 91

K**KEY**

beem-py-set command line option, 36
Key (class in beem.storage), 122
keyMap (beem.wallet.Wallet attribute), 131
keys (beem.wallet.Wallet attribute), 131
keyStorage (beem.wallet.Wallet attribute), 131

L

list_operations() (beem.transactionbuilder.TransactionBuilder method), 126
listen() (beem.notify.Notify method), 119
ListWitnesses (class in beem.witness), 132
lock() (beem.wallet.Wallet method), 131
locked() (beem.wallet.Wallet method), 131
log (in module beemgraphenebase.base58), 147

M

make_patch() (in module beem.utils), 127
market (beem.price.Price attribute), 121
Market (class in beem.market), 110
market_history() (beem.market.Market method), 112
market_history_buckets() (beem.market.Market method), 112
MasterPassword (beem.wallet.Wallet attribute), 129
masterpassword (beem.wallet.Wallet attribute), 131
MasterPassword (class in beem.storage), 123
me() (beem.steemconnect.SteemConnect method), 90
MEMO

beem-py-transfer command line option, 38
Memo (class in beem.memo), 116
Memo (class in beembase.objects), 142
Message (class in beem.message), 118
MissingKeyError, 108
MissingRequiredActiveAuthority, 135
mkdir_p() (beem.storage.DataDir method), 122
move_current_node_to_front() (beem.steem.Steem method), 84

MUTE

beem-py-mute command line option, 29
mute() (beem.account.Account method), 72

N**NAME**

beempy-addtoken command line option, 19
 beempy-deltoken command line option, 25
 name (beem.account.Account attribute), 72
 new_chart() (beem.asciichart.AsciiChart method), 77
 new_tx() (beem.steem.Steem method), 85
 newMaster() (beem.storage.MasterPassword method), 123
 newWallet() (beem.steem.Steem method), 85
 newWallet() (beem.wallet.Wallet method), 131
 next() (beemapi.graphenerpc.GrapheneRPC method), 139
 next() (beemapi.node.Nodes method), 138
 next_sequence() (beemgraphenebase.account.BrainKey method), 145
 NoAccessApi, 135
 NoApiWithName, 135
 node (beemapi.node.Nodes attribute), 138
 Node (class in beemapi.node), 138
 nodelist (beem.storage.Configuration attribute), 121
 NodeList (class in beem.nodelist), 88
 nodes (beem.storage.Configuration attribute), 121
 Nodes (class in beemapi.node), 138
 NoMethodWithName, 135
 normalize() (beemgraphenebase.account.BrainKey method), 145
 Notify (class in beem.notify), 118
 NoWalletException, 108
 NoWriteAccess, 108
 num_retries (beemapi.graphenerpc.GrapheneRPC attribute), 139
 num_retries_call (beemapi.graphenerpc.GrapheneRPC attribute), 139
 num_retries_call_reached (beemapi.node.Nodes attribute), 138
 NumRetriesReached, 135

O

object_type (in module beembase.objecttypes), 143
 object_type (in module beemgraphenebase.objecttypes), 148
 ObjectCache (class in beem.blockchainobject), 97
OBJECTS
 beempy-info command line option, 28
 OfflineHasNoRPCException, 108
 on_close() (beemapi.websocket.SteemWebsocket method), 137
 on_error() (beemapi.websocket.SteemWebsocket method), 137
 on_message() (beemapi.websocket.SteemWebsocket method), 137
 on_open() (beemapi.websocket.SteemWebsocket method), 137
 Operation (class in beembase.objects), 142
 Operation (class in beemgraphenebase.objects), 148
 operations (beem.block.Block attribute), 91

operations (in module beemgraphenebase.operationids), 148
 operations() (beembase.objects.Operation method), 142
 operations() (beemgraphenebase.objects.Operation method), 148
 ops (in module beembase.operationids), 143
 ops() (beem.blockchain.Blockchain method), 95
 ops_statistics() (beem.block.Block method), 91
 ops_statistics() (beem.blockchain.Blockchain method), 95
 Order (class in beem.price), 119
 orderbook() (beem.market.Market method), 113
ORDERID
 beempy-cancel command line option, 21

P

parent_author (beem.comment.Comment attribute), 101
 parent_permlink (beem.comment.Comment attribute), 101
 parse_time() (in module beem.utils), 127
 password (beem.storage.MasterPassword attribute), 123
 PasswordKey (class in beemgraphenebase.account), 145
 percent (beem.vote.Vote attribute), 128
 Permission (class in beembase.objects), 142
 permlink (beem.comment.Comment attribute), 101
 plot() (beem.asciichart.AsciiChart method), 77
 point() (beemgraphenebase.account.PublicKey method), 146
 Pool (class in beem.blockchain), 96
POST
 beempy-downvote command line option, 26
 beempy-upvote command line option, 40
 post() (beem.steem.Steem method), 85
 Post_discussions_by_payout (class in beem.discussions), 106
 precision (beem.asset.Asset attribute), 79
 prefix (beem.steem.Steem attribute), 86
 prefix (beem.wallet.Wallet attribute), 131
PRICE
 beempy-buy command line option, 21
 beempy-sell command line option, 36
 Price (class in beem.price), 119
 Price (class in beembase.objects), 142
 print_info() (beem.account.Account method), 72
 print_stats() (beem.vote.VotesObject method), 128
 print_summarize_table() (beem.account.AccountsObject method), 76
 printAsTable() (beem.account.AccountsObject method), 76
 printAsTable() (beem.vote.VotesObject method), 128
 printAsTable() (beem.witness.WitnessesObject method), 133
 PrivateKey (class in beemgraphenebase.account), 145
 process_block() (beem.notify.Notify method), 119

process_block() (beemapi.websocket.SteemWebsocket method), 137
profile (beem.account.Account attribute), 72
PUB
 beempy-delkey command line option, 24
PUB_SIGNING_KEY
 beempy-witnesscreate command line option, 42
PublicKey (class in beemgraphenebase.account), 146

Q

Query (class in beem.discussions), 106

R

recent_trades() (beem.market.Market method), 113
RecentByPath (class in beem.comment), 102
RecentReplies (class in beem.comment), 102
recover_with_latest_backup() (beem.storage.DataDir method), 122
refresh() (beem.account.Account method), 72
refresh() (beem.asset.Asset method), 79
refresh() (beem.block.Block method), 91
refresh() (beem.block.BlockHeader method), 92
refresh() (beem.comment.Comment method), 101
refresh() (beem.vote.Vote method), 128
refresh() (beem.witness.Witness method), 133
refresh_access_token() (beem.steemconnect.SteemConnect method), 90
refresh_data() (beem.steem.Steem method), 86
refreshBackup() (beem.storage.DataDir method), 122
remove_from_dict() (in module beem.utils), 127
removeAccount() (beem.wallet.Wallet method), 131
removePrivateKeyFromPublicKey() (beem.wallet.Wallet method), 132
removeTokenFromPublicKey() (beem.wallet.Wallet method), 132
rep (beem.account.Account attribute), 72
rep (beem.vote.Vote attribute), 128
Replies_by_last_update (class in beem.discussions), 107
reply() (beem.comment.Comment method), 101
reputation (beem.vote.Vote attribute), 128
reputation_to_score() (in module beem.utils), 127
request_send() (beemapi.graphenerpc.GrapheneRPC method), 139
reset_error_cnt() (beemapi.node.Nodes method), 138
reset_error_cnt_call() (beemapi.node.Nodes method), 138
reset_subscriptions() (beem.notify.Notify method), 119
reset_subscriptions() (beemapi.websocket.SteemWebsocket method), 137
resolve_authorperm() (in module beem.utils), 127
resolve_authorpermvoter() (in module beem.utils), 127
resolve_root_identifer() (in module beem.utils), 127
resteem() (beem.comment.Comment method), 101
results() (beem.blockchain.Pool method), 97

revoke_token() (beem.steemconnect.SteemConnect method), 90
reward (beem.comment.Comment attribute), 101
reward_balances (beem.account.Account attribute), 72
ripemd160() (in module beemgraphenebase.base58), 147
rpc (beem.wallet.Wallet attribute), 132
rpcclose() (beemapi.graphenerpc.GrapheneRPC method), 139
rpcconnect() (beemapi.graphenerpc.GrapheneRPC method), 139
RPCConnection, 135
RPCConnectionRequired, 109
RPCError, 135
RPCErrorDoRetry, 135
rpcexec() (beemapi.graphenerpc.GrapheneRPC method), 139
rpcexec() (beemapi.steemnodepc.SteemNodeRPC method), 135
rpcexec() (beemapi.websocket.SteemWebsocket method), 137
rpclogin() (beemapi.graphenerpc.GrapheneRPC method), 140
rshares (beem.vote.Vote attribute), 128
rshares_to_sbd() (beem.steem.Steem method), 86
rshares_to_vote_pct() (beem.steem.Steem method), 86
run() (beem.blockchain.Pool method), 97
run() (beem.blockchain.Worker method), 97
run_forever() (beemapi.websocket.SteemWebsocket method), 137

S

SaltException, 147
sanitize_permalink() (in module beem.utils), 127
saveEncryptedMaster() (beem.storage.MasterPassword method), 123
saving_balances (beem.account.Account attribute), 73
sbd (beem.vote.Vote attribute), 128
sell() (beem.market.Market method), 114
SessionInstance (class in beemapi.graphenerpc), 140
set_access_token() (beem.steemconnect.SteemConnect method), 90
set_cache_auto_clean() (beem.blockchainobject.BlockchainObject method), 97
set_cache_expiration() (beem.blockchainobject.BlockchainObject method), 97
set_default_account() (beem.steem.Steem method), 86
set_default_nodes() (beem.steem.Steem method), 86
set_default_vote_weight() (beem.steem.Steem method), 86
set_expiration() (beem.transactionbuilder.TransactionBuilder method), 126
set_next_node_on_empty_reply() (beemapi.steemnodepc.SteemNodeRPC method), 135

- set_parameter() (beem.asciichart.AsciiChart method), 77
 - set_password_storage() (beem.steem.Steem method), 87
 - set_session_instance() (in module beemapi.graphenerpc), 140
 - set_shared_config() (in module beem.instance), 110
 - set_shared_steem_instance() (in module beem.instance), 110
 - set_username() (beem.steemconnect.SteemConnect method), 90
 - set_withdraw_vesting_route() (beem.account.Account method), 73
 - setKeys() (beem.wallet.Wallet method), 132
 - setToken() (beem.wallet.Wallet method), 132
 - shared_session_instance() (in module beemapi.graphenerpc), 140
 - shared_steem_instance() (in module beem.instance), 110
 - SharedInstance (class in beem.instance), 110
 - sign() (beem.message.Message method), 118
 - sign() (beem.steem.Steem method), 87
 - sign() (beem.transactionbuilder.TransactionBuilder method), 126
 - sign() (beembase.signedtransactions.Signed_Transaction method), 143
 - sign() (beemgraphenebase.signedtransactions.Signed_Transaction method), 149
 - Signed_Transaction (class in beembase.signedtransactions), 143
 - Signed_Transaction (class in beemgraphenebase.signedtransactions), 148
 - SIGNING_KEY
 - beem-py-witnessenable command line option, 42
 - sleep_and_check_retries() (beemapi.node.Nodes method), 138
 - sp (beem.account.Account attribute), 73
 - sp_to_rshares() (beem.steem.Steem method), 87
 - sp_to_sbd() (beem.steem.Steem method), 87
 - sp_to_vests() (beem.steem.Steem method), 87
 - space_id (beem.blockchainobject.BlockchainObject attribute), 97
 - sqlDataBaseFile (beem.storage.DataDir attribute), 122
 - sqlite3_backup() (beem.storage.DataDir method), 122
 - sqlite3_copy() (beem.storage.DataDir method), 122
 - Steem (class in beem.steem), 79
 - steem_btc_ticker() (beem.market.Market static method), 115
 - steem_usd_implied() (beem.market.Market method), 115
 - SteemConnect (class in beem.steemconnect), 89
 - SteemNodeRPC (class in beemapi.steemnoderpc), 134
 - SteemWebsocket (class in beemapi.websocket), 136
 - stop() (beemapi.websocket.SteemWebsocket method), 137
 - storageDatabase (beem.storage.DataDir attribute), 122
 - str_to_bytes() (beem.aes.AESCipher static method), 76
 - stream() (beem.blockchain.Blockchain method), 95
 - suggest() (beemgraphenebase.account.BrainKey method), 145
 - symbol (beem.amount.Amount attribute), 79
 - symbol (beem.asset.Asset attribute), 79
 - symbols() (beem.price.Price method), 121
- ## T
- test_valid_objectid() (beem.blockchainobject.BlockchainObject method), 97
 - testid() (beem.blockchainobject.BlockchainObject method), 97
 - ticker() (beem.market.Market method), 115
 - time (beem.vote.Vote attribute), 128
 - time() (beem.block.Block method), 91
 - time() (beem.block.BlockHeader method), 92
 - time_elapsed() (beem.comment.Comment method), 101
 - TimeoutException, 135
 - title (beem.comment.Comment attribute), 101
 - TO
 - beem-py-powerdownroute command line option, 34
 - beem-py-transfer command line option, 38
 - toJson() (beemgraphenebase.objects.GrapheneObject method), 148
 - token (beem.wallet.Wallet attribute), 132
 - Token (class in beem.storage), 123
 - tokenStorage (beem.wallet.Wallet attribute), 132
 - total_balances (beem.account.Account attribute), 73
 - trade_history() (beem.market.Market method), 115
 - trades() (beem.market.Market method), 115
 - TransactionBuilder (class in beem.transactionbuilder), 124
 - transactions (beem.block.Block attribute), 91
 - transfer() (beem.account.Account method), 73
 - transfer_from_savings() (beem.account.Account method), 73
 - transfer_to_savings() (beem.account.Account method), 73
 - transfer_to_vesting() (beem.account.Account method), 74
 - Trending_tags (class in beem.discussions), 107
 - tryUnlockFromEnv() (beem.wallet.Wallet method), 132
 - tuple() (beem.amount.Amount method), 79
 - tx() (beem.steem.Steem method), 87
 - txbuffer (beem.steem.Steem attribute), 87
 - type_id (beem.account.Account attribute), 74
 - type_id (beem.asset.Asset attribute), 79
 - type_id (beem.blockchainobject.BlockchainObject attribute), 97
 - type_id (beem.comment.Comment attribute), 101
 - type_id (beem.vote.Vote attribute), 128
 - type_id (beem.witness.Witness attribute), 133
 - type_ids (beem.blockchainobject.BlockchainObject attribute), 97

U

UnauthorizedError, 135

unCompressed() (beem.graphenebase.account.PublicKey method), 146

UNFOLLOW

 beem.py-unfollow command line option, 39

unfollow() (beem.account.Account method), 74

UnhandledRPCError, 136

UnkownKey, 136

unlock() (beem.steem.Steem method), 87

unlock() (beem.wallet.Wallet method), 132

unlock_wallet() (beem.memo.Memo method), 118

unlocked() (beem.wallet.Wallet method), 132

UnnecessarySignatureDetected, 136

update() (beem.witness.Witness method), 133

update_account_metadata() (beem.account.Account method), 74

update_account_profile() (beem.account.Account method), 74

update_memo_key() (beem.account.Account method), 75

update_nodes() (beem.nodelist.NodeList method), 89

update_user_metadata() (beem.steemconnect.SteemConnect method), 90

updateToken() (beem.storage.Token method), 124

updateWif() (beem.storage.Key method), 123

upload() (beem.imageuploader.ImageUploader method), 109

upvote() (beem.comment.Comment method), 101

url (beemapi.node.Nodes attribute), 138

url_from_tx() (beem.steemconnect.SteemConnect method), 90

V

VALUE

 beem.py-set command line option, 36

 beem.py-setprofile command line option, 37

VARIABLE

 beem.py-delpoefile command line option, 25

 beem.py-setprofile command line option, 37

verify() (beem.message.Message method), 118

verify() (beembase.signedtransactions.Signed_Transaction method), 143

verify() (beemgraphenebase.signedtransactions.Signed_Transaction method), 149

verify_account_authority() (beem.account.Account method), 75

verify_authority() (beem.transactionbuilder.TransactionBuilder method), 126

version_string_to_int() (beemapi.graphenerpc.GrapheneRPC method), 140

VestingBalanceDoesNotExistsException, 109

vests_to_rshares() (beem.steem.Steem method), 87

vests_to_sbd() (beem.steem.Steem method), 88

vests_to_sp() (beem.steem.Steem method), 88

virtual_op_count() (beem.account.Account method), 75

volume24h() (beem.market.Market method), 116

Vote (class in beem.vote), 128

vote() (beem.comment.Comment method), 101

VOTE_WEIGHT

 beem.py-downvote command line option, 26

 beem.py-upvote command line option, 40

VoteDoesNotExistsException, 109

votee (beem.vote.Vote attribute), 128

voter (beem.vote.Vote attribute), 128

VotesObject (class in beem.vote), 128

VotingInvalidOnArchivedPost, 109

vp (beem.account.Account attribute), 75

W

wait_for_and_get_block() (beem.blockchain.Blockchain method), 96

Wallet (class in beem.wallet), 128

WalletExists, 109

WalletLocked, 109

weight (beem.vote.Vote attribute), 128

wipe() (beem.storage.Key method), 123

wipe() (beem.storage.MasterPassword static method), 123

wipe() (beem.storage.Token method), 124

wipe() (beem.wallet.Wallet method), 132

withdraw_vesting() (beem.account.Account method), 75

WITNESS

 beem.py-approvewitness command line option, 20

 beem.py-disapprovewitness command line option, 26

 beem.py-witness command line option, 41

 beem.py-witnesscreate command line option, 42

 beem.py-witnessdisable command line option, 42

 beem.py-witnessenable command line option, 42

 beem.py-witnessfeed command line option, 43

Witness (class in beem.witness), 132

witness_update() (beem.steem.Steem method), 88

WitnessDoesNotExistsException, 109

Witnesses (class in beem.witness), 133

WitnessesObject (class in beem.witness), 133

WitnessesRankedByVote (class in beem.witness), 133

WitnessesVotedByAccount (class in beem.witness), 134

WitnessProps (class in beembase.objects), 142

Worker (class in beem.blockchain), 97

working_nodes_count (beemapi.node.Nodes attribute), 138

WorkingNodeMissing, 136

WrongMasterPasswordException, 109

WrongMemoKey, 109

ws_send() (beemapi.graphenerpc.GrapheneRPC method), 140