

---

# **beem Documentation**

***Release 0.1***

**Holger Nahrstaedt**

**May 09, 2018**



---

## Contents

---

<b>1</b>	<b>About this Library</b>	<b>3</b>
<b>2</b>	<b>Quickstart</b>	<b>5</b>
<b>3</b>	<b>General</b>	<b>7</b>
<b>4</b>	<b>Packages</b>	<b>13</b>
<b>5</b>	<b>Glossary</b>	<b>67</b>
<b>6</b>	<b>Indices and tables</b>	<b>69</b>
	<b>Python Module Index</b>	<b>71</b>



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

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

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

### 3.1 Installation

Warning: install beem will install pycryptodome which is not compatible to pycrypto which is need for python-steem. At the moment, either beem or steem can be install at one maschine!

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
```

Install beem by pip:

```
pip install -U beem
```

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
```

### 3.1.1 Manual installation:

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

### 3.1.2 Upgrade

```
$ pip install --user --upgrade
```

## 3.2 Quickstart

## 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.

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

testnet = Steem(
    nobroadcast=True,
    bundle=True,
)

account = Account("test", steem_instance=testnet)
account.steem.wallet.unlock("supersecret")

account.transfer("test1", 1, "STEEM", account="test")
account.transfer("test1", 1, "STEEM", account="test")
account.transfer("test1", 1, "STEEM", account="test")
account.transfer("test1", 1, "STEEM", account="test")

pprint(testnet.broadcast())
```

### 3.3.2 Simple Sell Script

```

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

#
# Instantiate Steem (pick network via API node)
#
steem = Steem(
    nobroadcast=True    # <--- set this to False when you want to fire!
)

#
# Unlock the Wallet
#
steem.wallet.unlock("<supersecret>")

#
# 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_instance=steem
)

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

```

### 3.3.3 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

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

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

if __name__ == "__main__":

```

(continues on next page)

(continued from previous page)

```
#
# Instanciate Steem (pick network via API node)
#
steem = Steem(
    nobroadcast=True    # <--- set this to False when you want to fire!
)

#
# Unlock the Wallet
#
steem.wallet.unlock("<supersecret>")

#
# 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_instance=steem
)

sell()
```

## 3.4 Configuration

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

- API node URL
- default account name
- the encrypted master password

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 `steem.Steem` class, you can load the configuration directly by using:

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

### 3.4.1 API

**class** `beem.storage.Configuration`

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

**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

**nodes = ['wss://steemd.pevo.science', 'wss://gtg.steem.house:8090', 'wss://rpc.steemli**

Default configuration

## 3.5 Contributing to python-steem

We welcome your contributions to our project.

### 3.5.1 Repository

The *main* repository of python-steem is currently located at:

<https://github.com/holgern/beem>

### 3.5.2 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

### 3.5.3 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

### 3.5.4 Issues

Feel free to submit issues and enhancement requests.

### 3.5.5 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!

### 3.5.6 Copyright and Licensing

This library is open sources under the MIT license. We require your to release your code under that license as well.

## 3.6 Support and Questions

We have currently not setup a distinct channel for development around pysteemi. However, many of the contributors are frequently reading through these channels:



## 4.1 beem

### 4.1.1 beem package

#### Submodules

#### beem.account module

**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 it's corresponding functions.

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

---

**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 `active`)
- **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

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

Approve a witness

#### Parameters

- **witnesses** (*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 `steem.amount.Amount`.

#### balances

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

Cancel a withdrawal from ‘savings’ account. :param *str request\_id*: Identifier for tracking or cancelling the withdrawal :param *str account*: (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*)

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*. Args:

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

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

Convert SteemDollars to Steem (takes one week to settle) :param float *amount*: number of VESTS to withdraw :param *str account*: (optional) the source account for the transfer if not `default_account` :param *str request\_id*: (optional) identifier for tracking the conversion‘

**curation\_stats()**

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

Delegate SP to another account. Args:

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

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

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

**ensure\_full()**

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

Follow another account's blog :param str follow: Follow this account :param list what: List of states to follow

(defaults to *['blog']*)

**Parameters account** (*str*) – (optional) the account to allow access to (defaults to *default\_account*)

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

Returns limit similar accounts with name as array

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

Returns a generator for individual account transactions. This call can be used in a *for* loop. :param int index: first number of transactions to

return

#### Parameters

- **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*)
- **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*)
- **1) order** ((-1,)) – 1 for chronological, -1 for reverse order

**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 `steem.amount.Amount`.

**get\_balances** ()

**get\_bandwidth** (*bandwidth\_type=1, account=None, raw\_data=False*)

get\_account\_bandwidth

**get\_blog** (*entryId=0, limit=100, raw\_data=False, account=None*)

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

**get\_blog\_entries** (*entryId=0, limit=100, raw\_data=False, account=None*)

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

get\_owner\_history

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

Returns the curation reward of the last *days* days

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

**get\_feed** (*entryId=0, limit=100, raw\_data=False, account=None*)

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

**get\_followers** (*raw\_data=True*)

Returns the account followers as list

**get\_following** (*raw\_data=True*)

Returns who the account is following as list

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

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

Returns the account voting power recharge time in minutes

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

Returns the account recharge time

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

Returns the account voting power recharge time as timedelta object

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

**get\_reputation** ()

Returns the account reputation

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

Returns the account steem power

**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

**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 withdraw\_routes

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

Returns if the account has already voted for comment

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

**history** (*start=None, stop=None, 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*)
- **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*)

**history\_reverse** (*start=None, stop=None, 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*)
- **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*)

**interest** ()

Caluclate interest for an account :param str account: Account name to get interest for

**is\_fully\_loaded**

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

**name**

Returns the account name

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

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**

**sp**

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

Withdraw SBD or STEEM from 'savings' account. :param float amount: STEEM or SBD amount :param float asset: 'STEEM' or 'SBD' :param str memo: (optional) Memo :param str request\_id: (optional) identifier for tracking or cancelling the withdrawal :param str to: (optional) the source account for the transfer if not `default_account` :param str account: (optional) the source account for the transfer if not `default_account`

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

Transfer SBD or STEEM into a 'savings' account. :param float amount: STEEM or SBD amount :param float asset: 'STEEM' or 'SBD' :param str memo: (optional) Memo :param str to: (optional) the source account for the transfer if not `default_account` :param str account: (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, what=['blog'], account=None*)

Unfollow another account's blog :param str unfollow: Follow this account :param list what: List of states to follow

(defaults to ['blog'])

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

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

Update an account's meta data (*json\_meta*) :param dict *json*: The meta data to use (i.e. use Profile() from *account.py*)

**Parameters** *account* (*str*) – (optional) the account to allow access to (defaults to *default\_account*)

**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*)

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

Returns the number of individual account transactions

**vp**

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

Withdraw VESTS from the vesting account. :param float *amount*: number of VESTS to withdraw over a period of 104 weeks :param str *account*: (optional) the source account for the transfer if not *default\_account*

## beem.aes module

**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.amount module

**class** *beem.amount.Amount* (*amount*, *asset=None*, *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
```

**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 module

**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

**Return type** dict

---

**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 module

```
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*)
- **debug** (*bool*) – Enable Debugging (*optional*)
- **keys** (*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*)
- **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 (*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.

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 `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.info())
```

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** ()

**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. However, it **does not import the owner key** for security reasons. 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* pay a fee of this amount, and delegate the rest with VESTS (calculated automatically). Minimum: 1 STEEM. If left to 0 (Default), full fee is paid without VESTS delegation.
- **creator** (*str*) – which account should pay the registration fee (defaults to `default_account`)

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

**custom\_json** (*id*, *json\_data*, *required\_auths*=[], *required\_posting\_auths*=[])

Create a custom json operation :param str id: identifier for the custom json (max length 32 bytes) :param json json\_data: the json data to put into the custom\_json operation

### Parameters

- **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** ()

Returns the block interval in seconds

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

Return witness elected chain properties

::

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

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

Returns internal chain configuration.

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

This call returns the *dynamic global properties* :param bool use\_stored\_data: 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 :param bool use\_stored\_data: 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** ()

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

**get\_sbd\_per\_rshares** ()

Returns the current rshares to SBD ratio

**get\_steem\_per\_mvest** (*time\_stamp=None*)

Returns the current mvest to steem ratio

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

Return witness elected chain properties

**info** ()

Returns the global properties

**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

**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

**param bool force\_refresh** if True, data are forced to refreshed

**param float data\_refresh\_time\_seconds** set a new minimal refresh time in seconds

**register\_apis** (*apis=['network\_broadcast', 'account\_by\_key', 'follow', 'market\_history']*)

**rshares\_to\_sbd** (*rshares*)

Calculates the SBD amount of a vote

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

Set the default account to be used

**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** (*sp, voting\_power=10000, vote\_pct=10000*)

Obtain the r-shares :param number sp: Steem Power :param int voting\_power: voting power (100% = 10000) :param int vote\_pct: voting participation (100% = 10000)

**sp\_to\_sbd** (*sp, voting\_power=10000, vote\_pct=10000*)

**sp\_to\_vests** (*sp, timestamp=None*)

**tx** ()

Returns the default transaction buffer

**txbuffer**

Returns the currently active tx buffer

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

Unlock the internal wallet

**vests\_to\_sp** (*vests, timestamp=None*)

## beem.block module

```
class beem.block.Block (data, klass=None, space_id=1, object_id=None, lazy=False,
                        use_cache=True, id_item=None, steem_instance=None, *args, **kwargs)
```

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

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

Additionally 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)
```

---

**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

**ops** ()

Returns all block operations

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

Retuns a statistic with the occurance 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

```
class beem.block.BlockHeader(data, klass=None, space_id=1, object_id=None, lazy=False,
                             use_cache=True, id_item=None, steem_instance=None, *args,
                             **kwargs)
```

Bases: beem.blockchainobject.BlockchainObject

**block\_num**

Retuns the block number

**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 module

```
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) – (default) 3 maximum wait time for next block is max\_block\_wait\_repetition \* block\_interval

This class let's you deal with blockchain related data and methods. Read blockchain related data: .. code-block:: python

```
from beem.blockchain import Blockchain chain = Blockchain()
```

Read current block and blockchain info .. code-block:: python

```
print(chain.get_current_block()) print(chain.steem.info())
```

Monitor for new blocks .. .. code-block:: python

```
for block in chain.blocks(): print(block)
```

or each operation individually: .. code-block:: python

```
for operations in chain.ops(): print(operations)
```

**awaitTxConfirmation** (transaction, limit=10)

Returns the transaction as seen by the blockchain after being included into a block

---

**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.

**Parameters** **block\_num** (*int*) – Block number

**blocks** (*start=None, stop=None, max\_batch\_size=None, threading=False, thread\_num=8*)

Yields blocks starting from *start*.

**Parameters**

- **start** (*int*) – Starting block
- **stop** (*int*) – Stop at this block
- **mode** (*str*) – We here have the choice between “head” (the last block) and “irreversible” (the block that is confirmed by 2/3 of all block producers and is thus irreversible)

**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** ()

This call returns the current block

---

**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.

---

**static hash\_op** (*event*)

This method generates a hash of blockchain operation.



**is\_irreversible\_mode()**

**ops** (*start=None, stop=None, \*\*kwargs*)

Yields all operations (including virtual operations) starting from *start*.

#### Parameters

- **start** (*int*) – Starting block
- **stop** (*int*) – Stop at this block
- **mode** (*str*) – We here have the choice between “head” (the last block) and “irreversible” (the block that is confirmed by 2/3 of all block producers and is thus irreversible)
- **only\_virtual\_ops** (*bool*) – Only yield virtual operations

This call returns a list that only carries one operation and its type!

**ops\_statistics** (*start, stop=None, add\_to\_ops\_stat=None, verbose=False*)

Generates a 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

:param dict *add\_to\_ops\_stat*, if set, the result is added to *add\_to\_ops\_stat* :param bool *verbose*, if True, the current block number and timestamp is printed This call returns a dict with all possible operations and their occurrence.

**stream** (*opNames=[], \*args, \*\*kwargs*)

Yield specific operations (e.g. comments) only

#### Parameters

- **opNames** (*array*) – List of operations to filter for
- **start** (*int*) – Start at this block
- **stop** (*int*) – Stop at this block
- **mode** (*str*) – We here have the choice between “head” (the last block) and “irreversible” (the block that is confirmed by 2/3 of all block producers and is thus irreversible)

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 key depend on the actualy operation.

**wait\_for\_and\_get\_block** (*block\_number, blocks\_waiting\_for=None*)

Get the desired block from the chain, if the current head block is smaller (for both head and irreversible) then we wait, but a maximum of *blocks\_waiting\_for \* max\_block\_wait\_repetition* time before failure. :param int *block\_number*: desired block number :param int *blocks\_waiting\_for*: (default) difference between *block\_number* and current head

how many blocks we are willing to wait, positive int

## beem.comment module

**class** `beem.comment.Comment` (*authorperm, full=False, lazy=False, steem\_instance=None*)

Bases: `beem.blockchainobject.BlockchainObject`

Read data about a Comment/Post in the chain

#### Parameters

- **authorperm** (*str*) – perm link to post/comment
- **steem\_instance** (*steem*) – Steem() instance to use when accessing a RPC

**author**

**authorperm**

**body**

**category**

**comment\_options** (*options, identifier=None, account=None*)

Set the comment options :param str identifier: Post identifier :param dict options: The options to define.  
:param str account: (optional) the account to allow access

to (defaults to default\_account)

**For the options, you have these defaults:::**

```
{ "author": "", "permink": "", "max_accepted_payout": "1000000.000 SBD", "per-  
cent_steem_dollars": 10000, "allow_votes": True, "allow_curation_rewards": True,  
}
```

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

Delete an existing post/comment :param str identifier: Identifier for the post to upvote Takes  
the form @author/permink

**Parameters account** (*str*) – Voter to use for voting. (Optional)

If voter is not defines, the default\_account will be taken or a ValueError will be raised

**downvote** (*weight=-100, voter=None*)

Downvote the post :param float weight: (optional) Weight for posting (-100.0 - +100.0) defaults to -100.0  
:param str voter: (optional) Voting account

**edit** (*body, meta=None, replace=False*)

Edit an existing post :param str body: Body of the reply :param json meta: JSON meta object that can be  
attached to the  
post. (optional)

**Parameters replace** (*bool*) – Instead of calculating a *diff*, replace the post entirely (defaults  
to False)

**get\_reblogged\_by** (*identifier=None*)

**get\_votes** ()

**id**

**is\_comment** ()

Retuns True if post is a comment

**is\_main\_post** ()

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

**json** ()

**json\_metadata**

**parent\_author**

**parent\_permlink**

**permlink**

**post** (*title=None, body=None, author=None, permlink=None, reply\_identifier=None, json\_metadata=None, comment\_options=None, community=None, tags=None, beneficiaries=None, self\_vote=False*)

Create a new post. If this post is intended as a reply/comment, *reply\_identifier* needs to be set with the identifier of the parent post/comment (eg. *@author/permlink*). Optionally you can also set *json\_metadata*, *comment\_options* and upvote the newly created post as an author. Setting category, tags or community will override the values provided in *json\_metadata* and/or *comment\_options* where appropriate. Args: *title* (str): Title of the post *body* (str): Body of the post/comment *author* (str): Account are you posting from *permlink* (str): Manually set the permlink (defaults to None).

If left empty, it will be derived from title automatically.

**reply\_identifier (str): Identifier of the parent post/comment (only** if this post is a reply/comment).

**json\_metadata (str, dict): JSON meta object that can be attached to** the post.

**comment\_options (str, dict): JSON options object that can be** attached to the post.

**Example::**

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

**community (str): (Optional) Name of the community we are posting** into. This will also override the community specified in *json\_metadata*.

**tags (str, list): (Optional) A list of tags (5 max) to go with the** post. This will also override the tags specified in *json\_metadata*. The first tag will be used as a 'category'. If provided as a string, it should be space separated.

**beneficiaries (list of dicts): (Optional) A list of beneficiaries** for posting reward distribution. This argument overrides beneficiaries as specified in *comment\_options*.

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

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

**self\_vote (bool): (Optional) Upvote the post as author, right after** posting.

**refresh()**

**reply** (*body, title="", author="", meta=None*)

Reply to an existing post :param str body: Body of the reply :param str title: Title of the reply post :param str author: Author of reply (optional) if not provided

*default\_user* will be used, if present, else a *ValueError* will be raised.

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

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

Resteem a post :param str identifier: post identifier (@<account>/<permlink>) :param str account: (optional) the account to allow access

to (defaults to default\_account)

**title**

**type\_id** = 8

**upvote** (*weight=100, voter=None*)

Upvote the post :param float weight: (optional) Weight for posting (-100.0 - +100.0) defaults to +100.0 :param str voter: (optional) Voting account

**vote** (*weight, account=None, identifier=None, \*\*kwargs*)

Vote for a post :param str identifier: Identifier for the post to upvote Takes the form @author/permlink

#### Parameters

- **weight** (*float*) – Voting weight. Range: -100.0 - +100.0. May not be 0.0
- **account** (*str*) – Voter to use for voting. (Optional)

If voter is not defines, the default\_account will be taken or a ValueError will be raised

**class** beem.comment.**RecentByPath** (*path='promoted', category=None, 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 accesing a RPC

**class** beem.comment.**RecentReplies** (*author, skip\_own=True, steem\_instance=None*)

Bases: list

Obtain a list of recent replies

#### Parameters

- **author** (*str*) – author
- **steem\_instance** (*steem*) – Steem() instance to use when accesing a RPC

## beem.discussions module

**class** beem.discussions.**Comment\_discussions\_by\_payout** (*discussion\_query, steem\_instance=None*)

Bases: list

get\_comment\_discussions\_by\_payout

:param str discussion\_query :param steem steem\_instance: Steem() instance to use when accesing a RPC

```
class beem.discussions.Discussions_by_active (discussion_query, steem_instance=None)
    Bases: list

    get_discussions_by_active

    :param str discussion_query :param steem steem_instance: Steem() instance to use when accesing a RPC

class beem.discussions.Discussions_by_blog (discussion_query, steem_instance=None)
    Bases: list

    get_discussions_by_blog

    :param str discussion_query, tag musst be set to a username :param steem steem_instance: Steem() instance to
    use when accesing a RPC

class beem.discussions.Discussions_by_cashout (discussion_query,
                                                steem_instance=None)
    Bases: list

    get_discussions_by_cashout. This query seems to be broken at the moment. The output is always empty.

    :param str discussion_query :param steem steem_instance: Steem() instance to use when accesing a RPC

class beem.discussions.Discussions_by_children (discussion_query,
                                                steem_instance=None)
    Bases: list

    get_discussions_by_children

    :param str discussion_query :param steem steem_instance: Steem() instance to use when accesing a RPC

class beem.discussions.Discussions_by_comments (discussion_query,
                                                steem_instance=None)
    Bases: list

    get_discussions_by_comments

    :param str discussion_query :param steem steem_instance: Steem() instance to use when accesing a RPC

class beem.discussions.Discussions_by_created (discussion_query,
                                                steem_instance=None)
    Bases: list

    get_discussions_by_created

    :param str discussion_query :param steem steem_instance: Steem() instance to use when accesing a RPC

class beem.discussions.Discussions_by_feed (discussion_query, steem_instance=None)
    Bases: list

    get_discussions_by_feed

    :param str discussion_query, tag musst be set to a username :param steem steem_instance: Steem() instance to
    use when accesing a RPC

class beem.discussions.Discussions_by_hot (discussion_query, steem_instance=None)
    Bases: list

    get_discussions_by_hot

    :param str discussion_query :param steem steem_instance: Steem() instance to use when accesing a RPC

class beem.discussions.Discussions_by_promoted (discussion_query,
                                                steem_instance=None)
    Bases: list

    get_discussions_by_promoted
```

```
:param str discussion_query :param steem steem_instance: Steem() instance to use when accesing a RPC
```

```
class beem.discussions.Discussions_by_trending (discussion_query,
                                              steem_instance=None)
    Bases: list
    get_discussions_by_trending

:param Query discussion_query :param steem steem_instance: Steem() instance to use when accesing a RPC
```

```
class beem.discussions.Discussions_by_votes (discussion_query, steem_instance=None)
    Bases: list
    get_discussions_by_votes

:param str discussion_query :param steem steem_instance: Steem() instance to use when accesing a RPC
```

```
class beem.discussions.Post_discussions_by_payout (discussion_query,
                                                  steem_instance=None)
    Bases: list
    get_post_discussions_by_payout

:param str discussion_query :param steem steem_instance: Steem() instance to use when accesing a RPC
```

```
class beem.discussions.Query (limit=0,    tag="",    truncate_body=0,    filter_tags=[],    se-
                             lect_authors=[],    select_tags=[],    start_author=None,
                             start_permlink=None,    parent_author=None,    par-
                             ent_permlink=None)
    Bases: dict

:param int limit :param str tag :param int truncate_body :param array filter_tags :param array select_authors
:param array select_tags :param str start_author :param str start_permlink :param str parent_author :param str
parent_permlink
```

## beem.exceptions module

```
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.BlockDoesNotExistsException
```

Bases: Exception

The block does not exist

```
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.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.KeyNotFound`

Bases: `Exception`

Key not found

**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 `peerplays.wallet.create()` to create a new wallet

**exception** `beem.exceptions.ObjectNotInProposalBuffer`

Bases: `Exception`

Object was not found in proposal

**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

## beem.market module

**class** `beem.market.Market` (*steem\_instance=None*)

Bases: `dict`

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

### Parameters

- **steem\_instance** (*steem.steem.Steem*) – Steem instance
- **base** (*steem.asset.Asset*) – Base asset
- **quote** (*steem.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 it's 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 `steem.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. `USD:BTS` with `USD` being the `quote`), while the `base` only refers to a secondary asset for a trade. This means, if you call `steem.market.Market.sell()` or `steem.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

**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 USD\_BTS market is priced in BTS per USD.

**Example:** in the USD\_BTS market, a price of 300 means a USD is worth 300 BTS

**Note:** All prices returned are in the **reversed** orientation as the market. I.e. in the BTC/BTS market, prices are BTS per BTC. 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 USD for 100 BTS/USD
- This means that you actually place a sell order for 1000 BTS in order to obtain **at least** 10 USD
- If an order on the market exists that sells USD for cheaper, you will end up with more than 10 USD

**cancel** (*orderNumbers*, *account=None*, *\*\*kwargs*)

Cancels an order you have placed in a given market. Requires only the “orderNumbers”. An order number takes the form 1.7.xxx. :param str orderNumbers: The Order Object id of the form 1.7.xxxx

**get\_string** (*separator=':'*)

Return a formatted string that identifies the market, e.g. USD:BTS

**Parameters separator** (*str*) – The separator of the assets (defaults to :)

**market\_history** (*bucket\_seconds=300*, *start\_age=3600*, *end\_age=0*)

**market\_history\_buckets** ()

**orderbook** (*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. :param int limit: Limit the amount of orders (default: 25) Sample output: .. code-block:: js

```
{'bids': [0.003679 USD/BTS (1.9103 USD|519.29602 BTS), 0.003676 USD/BTS (299.9997
USD|81606.16394 BTS), 0.003665 USD/BTS (288.4618 USD|78706.21881 BTS), 0.003665
USD/BTS (3.5285 USD|962.74409 BTS), 0.003665 USD/BTS (72.5474 USD|19794.41299
BTS)], 'asks': [0.003738 USD/BTS (36.4715 USD|9756.17339 BTS), 0.003738 USD/BTS
(18.6915 USD|5000.00000 BTS), 0.003742 USD/BTS (182.6881 USD|48820.22081 BTS),
0.003772 USD/BTS (4.5200 USD|1198.14798 BTS), 0.003799 USD/BTS (148.4975
USD|39086.59741 BTS)]}
```

---

**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

---

**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)

Sample output:

```
{'bids': [0.003679 USD/BTS (1.9103 USD|519.29602 BTS),
0.003676 USD/BTS (299.9997 USD|81606.16394 BTS),
0.003665 USD/BTS (288.4618 USD|78706.21881 BTS),
0.003665 USD/BTS (3.5285 USD|962.74409 BTS),
0.003665 USD/BTS (72.5474 USD|19794.41299 BTS)],
'asks': [0.003738 USD/BTS (36.4715 USD|9756.17339 BTS),
0.003738 USD/BTS (18.6915 USD|5000.00000 BTS),
0.003742 USD/BTS (182.6881 USD|48820.22081 BTS),
0.003772 USD/BTS (4.5200 USD|1198.14798 BTS),
0.003799 USD/BTS (148.4975 USD|39086.59741 BTS)]}
```

---

**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 USD\_BTS market is priced in BTS per USD.

**Example:** in the USD\_BTS market, a price of 300 means a USD is worth 300 BTS

---

**Note:** All prices returned are in the **reversed** orientation as the market. I.e. in the BTC/BTS market, prices are BTS per BTC. That way you can multiply prices with *1.05* to get a +5%.

---

**ticker** (*raw\_data=False*)

Returns the ticker for all markets.

Output Parameters:

- **last**: Price of the order last filled

- **lowestAsk**: Price of the lowest ask
- **highestBid**: Price of the highest bid
- **baseVolume**: Volume of the base asset
- **quoteVolume**: Volume of the quote asset
- **percentChange**: 24h change percentage (in %)
- **settlement\_price**: Settlement Price for borrow/settlement
- **core\_exchange\_rate**: Core exchange rate for payment of fee in non-BTS asset
- **price24h**: the price 24h ago

Sample Output:

```
{
  {
    "quoteVolume": 48328.73333,
    "quoteSettlement_price": 332.3344827586207,
    "lowestAsk": 340.0,
    "baseVolume": 144.1862,
    "percentChange": -1.9607843231354893,
    "highestBid": 334.20000000000005,
    "latest": 333.33333330133934,
  }
}
```

**trades** (*limit=25, start=None, stop=None, raw\_data=False*)

Returns your trade history for a given market.

#### Parameters

- **limit** (*int*) – Limit the amount of orders (default: 25)
- **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:

```
{
  "BTS": 361666.63617,
  "USD": 1087.0
}
```

## beem.memo module

**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.

In BitShares, memos are AES-256 encrypted with a shared secret between sender and receiver. It is derived from the memo private key of the sender and the memo public key of the receiver.

In order for the receiver to decode the memo, the shared secret has to be derived from the receiver's private key and the senders public key.

The memo public key is part of the account and can be retrieved with the `get_account` call:

```
get_account <accountname>
{
  [...]
  "options": {
    "memo_key": "GPH5TPTziKkLexhVKsQKtSpo4bAv5RnB8oXcG4sMHEwCcTf3r7dqE",
    [...]
  },
  [...]
}
```

while the memo private key can be dumped with `dump_private_keys`

The 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.

The high level memo class makes use of the `pysteam` 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)

```

```

from getpass import getpass
from beem.block import Block
from beem.memo import Memo

# Obtain a transfer from the blockchain
block = Block(23755086)           # block
transaction = block["transactions"][3]  # transactions
op = transaction["operations"][0]      # operation
op_id = op[0]                     # operation type
op_data = op[1]                   # operation payload

# Instantiate Memo for decoding
memo = Memo()

# Unlock wallet
memo.unlock_wallet(getpass())

# Decode memo
# Raises exception if required keys not available in the wallet
print(memo.decrypt(op_data["transfer"]))

```

**decrypt (memo)**

Decrypt a memo

**Parameters** **memo** (*str*) – encrypted memo message**Returns** encrypted memo**Return type** str**encrypt (memo, bts\_encrypt=False)**

Encrypt a memo

**Parameters** **memo** (*str*) – clear text memo message**Returns** encrypted memo**Return type** str**unlock\_wallet (\*args, \*\*kwargs)**

Unlock the library internal wallet

**beem.message module****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 module

**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** (*fn*) – 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 module

**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
0.662600000 SBD/STEEM
```

**as\_base** (*base*)

Returns the price instance so that the base asset is *base*.

Note: This makes a copy of the object!

**as\_quote** (*quote*)

Returns the price instance so that the quote asset is *quote*.

Note: This makes a copy of the object!

**copy** () → a shallow copy of D

**invert** ()

Invert the price (e.g. go from SBD/STEEM into STEEM/SBD)

**json** ()

**market**

Open the corresponding market

**Returns** Instance of `beem.market.Market` for the corresponding pair of assets.

**symbols** ()

## beem.storage module

**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** = {'node': ['wss://steemd.pevo.science', 'wss://gtg.steem.house:8090']}

**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** ()

**nodes** = ['wss://steemd.pevo.science', 'wss://gtg.steem.house:8090', 'wss://rpc.steemli

Default configuration

**class** beem.storage.DataDir

Bases: object

This class ensures that the user's data is stored in its OS preotected user directory:

**OSX:**

- ~/Library/Application Support/<AppName>

**Windows:**

- C:\Documents and Settings<User>\Application Data\Local Settings<AppAuthor>\<AppName>
- C:\Documents and Settings<User>\Application Data<AppAuthor>\<AppName>

**Linux:**

- ~/.local/share/<AppName>

Furthermore, it offers an interface to generated backups in the *backups/* directory every now and then.

**appauthor** = 'beem'

**appname** = 'beem'

**clean\_data** ()

Delete files older than 70 days

**data\_dir** = '/home/docs/.local/share/beem'

**mkdir\_p** ()

Ensure that the directory in which the data is stored exists

**refreshBackup** ()

Make a new backup

**sqlDataBaseFile** = '/home/docs/.local/share/beem/beem.sqlite'

**sqlite3\_backup** (*dbfile*, *backupdir*)

Create timestamped database copy

**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 = ''

saveEncryptedMaster()
    Store the encrypted master password in the configuration store

static wipe(sure=False)
    Remove all keys from configStorage
```

## beem.transactionbuilder module

```
class beem.transactionbuilder.TransactionBuilder(tx={}, proposer=None, expiration=None, steem_instance=None)
```

Bases: dict

This class simplifies the creation of transactions by adding operations and signers. To build your own transactions and sign them

```
from beem.transactionbuilder import TransactionBuilder
from beembase.operations import Transfer
tx = TransactionBuilder()
tx.appendOps(Transfer(**{
    "from": "test",
    "to": "test1",
    "amount": "1 STEEM",
    "memo": ""
}))
tx.appendSigner("test", "active")
tx.sign()
tx.broadcast()
```

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

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

FIXME: Does not work with owner keys!

**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

**appendWif** (*wif*)

Add a wif that should be used for signing of the transaction.

**broadcast** ()

Broadcast a transaction to the steem network

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

**clear** ()

Clear the transaction builder and start from scratch

**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

**is\_empty()**

**json()**

Show the transaction as plain json

**list\_operations()**

**set\_expiration(p)**

**sign()**

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.

**verify\_authority()**

Verify the authority of the signed transaction

## beem.utils module

**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:

**beem.utils.construct\_authorpermvoter(\*args)**

Create a vote identifier from vote object or arguments. Examples:

**beem.utils.derive\_permalink(title, parent\_permalink=None, parent\_author=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.resolve_authorperm (identifier)
    Correctly split a string containing an authorperm.

    Splits the string into author and perm link with the following separator: /.

beem.utils.resolve_authorpermvoter (identifier)
    Correctly split a string containing an authorpermvoter.

    Splits the string into author and perm link with the following separator: / and |.

beem.utils.resolve_root_identifier (url)

beem.utils.sanitize_permalink (permalink)

beem.utils.test_proposal_in_buffer (buf, operation_name, id)
```

## beem.vote module

```
class beem.vote.AccountVotes (account, 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, 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("theaussiegame/cryptokittie-giveaway-number-2|")
```

```
json()
percent
refresh()
rep
reputation
rshares
sbd
time
type_id = 11
voter
weight

class beem.vote.VotesObject
    Bases: list

    printAsTable (sort_key='sbd', reverse=True)
```

## beem.wallet module

```
class beem.wallet.Wallet (rpc=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.purgeWallet()
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

**changePassphrase** (*new\_pwd*)

Change the passphrase for the wallet database

**clear\_local\_keys** ()

Clear all manually provided keys

**configStorage = None**

**create** (*pwd*)

Alias for newWallet()

**created** ()

Do we have a wallet database already?

**decrypt\_wif** (*encwif*)

decrypt a wif key

**encrypt\_wif** (*wif*)

Encrypt a wif key

**getAccount** (*pub*)

Get the account data for a public key (first account found for this public key)

**getAccountFromPrivateKey** (*wif*)

Obtain account name from private key

**getAccountFromPublicKey** (*pub*)

Obtain the first account name from public key

**getAccounts** ()

Return all accounts installed in the wallet database

**getAccountsFromPublicKey** (*pub*)

Obtain all accounts associated with a public key

**getActiveKeyForAccount** (*name*)

Obtain owner Active Key for an account from the wallet database

**getAllAccounts** (*pub*)

Get the account data for a public key (all accounts found for this public key)

**getKeyType** (*account*, *pub*)

Get key type

**getMemoKeyForAccount** (*name*)

Obtain owner Memo Key for an account from the wallet database

**getOwnerKeyForAccount** (*name*)

Obtain owner Private Key for an account from the wallet database

**getPostingKeyForAccount** (*name*)

Obtain owner Posting Key for an account from the wallet database

**getPrivateKeyForPublicKey** (*pub*)

Obtain the private key for a given public key

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

**getPublicKeys** ()

Return all installed public keys

**keyMap** = {}

**keyStorage** = None

**keys** = {}

**lock** ()

Lock the wallet database

**locked** ()

Is the wallet database locked?

**masterpassword** = None

**newWallet** (*pwd*)

Create a new wallet database

**removeAccount** (*account*)

Remove all keys associated with a given account

**removePrivateKeyFromPublicKey** (*pub*)

Remove a key from the wallet database

**rpc** = None

**setKeys** (*loadkeys*)

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

**steem** = None

**tryUnlockFromEnv** ()

**unlock** (*pwd=None*)

Unlock the wallet database

**unlocked** ()

Is the wallet database unlocked?

**wipe** (*sure=False*)

Purge all data in wallet database



**beem.witness module**

**class** beem.witness.**ListWitnesses** (*from\_account, limit, steem\_instance=None*)

Bases: *beem.witness.WitnessesObject*

Obtain a list of witnesses which have been voted by an account

**Parameters**

- **from\_account** (*str*) – Account name
- **steem\_instance** (*steem*) – Steem() instance to use when accessing a RPC

**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")
```

**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"]

**refresh** ()

**type\_id** = 3

**update** (*signing\_key, url, props, account=None*)

Update witness :param pubkey signing\_key: Signing key :param str url: URL :param dict props: Properties :param str account: (optional) witness account name

**Properties:::**

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

**class** beem.witness.**Witnesses** (*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

**class** beem.witness.**WitnessesObject**

Bases: *list*

**printAsTable** (*sort\_key='votes', reverse=True*)

**class** beem.witness.**WitnessesRankedByVote** (*from\_account="", limit=100, steem\_instance=None*)

Bases: *beem.witness.WitnessesObject*

Obtain a list of witnesses ranked by Vote

**Parameters**

- **from\_account** (*str*) – Witness name
- **steem\_instance** (*steem*) – Steem() instance to use when accessing a RPC

**class** `beem.witness.WitnessesVotedByAccount` (*account*, *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

**Module contents**

beem.

## 4.2 beembase

### 4.2.1 beembase package

**Submodules****beembase.chains module****beembase.memo module**

`beembase.memo.decode_memo` (*priv*, *message*)

Decode a message with a shared secret between Alice and Bob :param PrivateKey priv: Private Key (of Bob)  
:param base58encoded message: Encrypted Memo message :return: Decrypted message :rtype: str :raise 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 :param PrivateKey priv: Private Key (of Alice)  
:param PublicKey pub: Public Key (of Bob) :param int nonce: Random nonce :param str message: Memo  
message :return: Encrypted message :rtype: 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 :param hex shared\_secret: Shared Secret to use as encryption key :param int nonce:  
Random nonce :return: AES instance and checksum of the encryption key :rtype: 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 module

`beembase.objects.AccountId(asset)`

**class** `beembase.objects.AccountOptions(*args, **kwargs)`

Bases: `beemgraphenebase.objects.GrapheneObject`

**class** `beembase.objects.Amount(d)`

Bases: `object`

```
beembase.objects.AssetId(asset)

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. Args:

        beneficiaries (list): A static_variant containing beneficiaries.

    Example:

        ::

        [0,
          {'beneficiaries': [ {'account': 'furiion', '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.ObjectId(object_str, type_verify=None)
    Bases: beemgraphenebase.types.ObjectId

    Encodes object/protocol ids

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 module

`beembase.objecttypes.object_type = {'account': 2, 'account_history': 18, 'block_summary': ...}`  
Object types for object ids

## beembase.operationids module

`beembase.operationids.getOperationNameForId(i)`  
Convert an operation id into the corresponding string

`beembase.operationids.ops = ['vote', 'comment', 'transfer', 'transfer_to_vesting', 'withdrawal']`  
Operation ids

## beembase.operations module

`beembase.operationids.getOperationNameForId(i)`  
Convert an operation id into the corresponding string

`beembase.operationids.ops = ['vote', 'comment', 'transfer', 'transfer_to_vesting', 'withdrawal']`  
Operation ids

## beembase.transactions module

### Module contents

`beembase.`

## 4.3 beemapi

### 4.3.1 beemapi package

#### Submodules

#### SteemNodeRPC

This class allows to call API methods exposed by the witness node via websockets.

#### Definition

**class** `beemapi.steemnode.rpc.SteemNodeRPC(*args, **kwargs)`

This class allows to call API methods exposed by the witness node via websockets / rpc-json.

**\_\_getattr\_\_** (*name*)

Map all methods to RPC calls and pass through the arguments.

**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

**beemapi.exceptions module**

**exception** beemapi.exceptions.**MissingRequiredActiveAuthority**

Bases: beemgrapheneapi.rpcutils.RPCError

**exception** beemapi.exceptions.**NoAccessApi**

Bases: beemgrapheneapi.rpcutils.RPCError

**exception** beemapi.exceptions.**NoApiWithName**

Bases: beemgrapheneapi.rpcutils.RPCError

**exception** beemapi.exceptions.**NoMethodWithName**

Bases: beemgrapheneapi.rpcutils.RPCError

**exception** beemapi.exceptions.**UnhandledRPCError**

Bases: beemgrapheneapi.rpcutils.RPCError

beemapi.exceptions.**decodeRPCErrorMsg**(e)

Helper function to decode the raised Exception and give it a python Exception class

**SteemWebsocket**

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()
```

**Defintion**

**class** beemapi.websocket.**SteemWebsocket**(urls, user="", password="", only\_block\_id=False, on\_block=None, keep\_alive=25, num\_retries=-1, \*args, \*\*kwargs)

Create a websocket connection and request push notifications

**Parameters**

- **urls** (str) – Either a single Websocket URL, or a list of URLs
- **user** (str) – Username for Authentication
- **password** (str) – Password for Authentication
- **keep\_alive** (int) – seconds between a ping to the backend (defaults to 25seconds)

After instantiating this class, you can add event slots for:

- `on_block`

which will be called accordingly with the notification message received from the Steem node:

```
ws = SteemWebsocket (
    "wss://gtg.steem.house:8090",
)
ws.on_block += print
ws.run_forever()
```

Notices:

- `on_block:`

```
'0062f19df70ecf3a478a84b4607d9ad8b3e3b607'
```

**`__SteemWebsocket__set_subscriptions()`**

set subscriptions of `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, *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

## Module contents

beemapi.

## 4.4 beemgraphenebase

### 4.4.1 beemgraphenebase package

#### Submodules

#### beemgraphenebase.account module

**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))

**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\_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.

**compressedpubkey()**

Derive uncompressed public 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

**point()**

Return the point for the public key

**unCompressed()**

Derive uncompressed key

**beemgraphenebase.base58 module**

**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 module

**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 module

## beemgraphenebase.objects module

**class** `beemgraphenebase.objects.GrapheneObject` (*data=None*)

Bases: `object`

Core abstraction class

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

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

`json()`

`toJson()`

**class** `beemgraphenebase.objects.Operation` (*op*)

Bases: `object`

`getOperationNameForId(i)`

Convert an operation id into the corresponding string

`operations()`

`beemgraphenebase.objects.isArgsThisClass(self, args)`

## beemgraphenebase.objecttypes module

`beemgraphenebase.objecttypes.object_type = {'OBJECT_TYPE_COUNT': 3, 'account': 2, 'base':`

`Object types for object ids`

## beemgraphenebase.operations module

`beemgraphenebase.operationids.operations = {'demooperation': 0}`

Operation ids

## beemgraphenebase.transactions module

`beemgraphenebase.transactions.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`

`beemgraphenebase.transactions.getBlockParams(ws)`

Auxiliary method to obtain `ref_block_num` and `ref_block_prefix`. Requires a websocket connection to a witness node!

## Module contents

`beemgraphenebase.`

## 4.5 beemgrapheneapi

### 4.5.1 beemgrapheneapi package

#### Submodules

#### 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.

**class** `beemgrapheneapi.graphenerpc.GrapheneRPC(urls, user=None, password=None, **kwargs)`

This class allows to call API methods synchronously, without callbacks.

It logs in and registers to the APIs:

- `database`
- `history`

#### Parameters

- **urls** (*str*) – Either a single Websocket/Http URL, or a list of URLs
- **user** (*str*) – Username for Authentication
- **password** (*str*) – Password for Authentication
- **apis** (*Array*) – List of APIs to register to (default: [`“database”`, `“network_broadcast”`])
- **num\_retries** (*int*) – Try  $x$  times to num\_retries to a node on disconnect, -1 for indefinitely

Available APIs

- database
- network\_node
- network\_broadcast

Usage:

---

**Note:** This class allows to call methods available via websocket. If you want to use the notification subsystem, please use `GrapheneWebsocket` instead.

---

**get\_request\_id()**

Get request id.

**is\_appbase\_ready()**

Check if node is appbase ready

**register\_apis()**

Register apis.

**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
- **RPCError** – if the server returns an error

**rpclogin** (*user, password*)

Login into Websocket

## Module contents

beemgrapheneapi.

## CHAPTER 5

---

Glossary

---





## CHAPTER 6

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`



### b

- beem, 54
- beem.account, 13
- beem.aes, 19
- beem.amount, 19
- beem.asset, 20
- beem.block, 26
- beem.blockchain, 27
- beem.comment, 29
- beem.discussions, 32
- beem.exceptions, 34
- beem.market, 36
- beem.memo, 39
- beem.message, 41
- beem.notify, 42
- beem.price, 42
- beem.steem, 21
- beem.storage, 44
- beem.transactionbuilder, 47
- beem.utils, 48
- beem.vote, 49
- beem.wallet, 50
- beem.witness, 53
- beemapi, 60
- beemapi.exceptions, 58
- beembase, 57
- beembase.chains, 54
- beembase.memo, 54
- beembase.objects, 55
- beembase.objecttypes, 57
- beembase.operationids, 57
- beembase.transactions, 57
- beemgrapheneapi, 66
- beemgraphenebase, 65
- beemgraphenebase.account, 60
- beemgraphenebase.base58, 62
- beemgraphenebase.bip38, 63
- beemgraphenebase.objects, 64
- beemgraphenebase.objecttypes, 64
- beemgraphenebase.operationids, 64
- beemgraphenebase.transactions, 65



## Symbols

`__SteemWebsocket__set_subscriptions()`  
 (beemapi.websocket.SteemWebsocket  
 method), 59  
`__events__` (beemapi.websocket.SteemWebsocket at-  
 tribute), 59  
`__getattr__()` (beemapi.steemnodepc.SteemNodeRPC  
 method), 57  
`__getattr__()` (beemapi.websocket.SteemWebsocket  
 method), 59  
`__init__()` (beemapi.websocket.SteemWebsocket  
 method), 59  
`__module__` (beemapi.websocket.SteemWebsocket at-  
 tribute), 59  
`_ping()` (beemapi.websocket.SteemWebsocket method),  
 59

## A

account (beem.witness.Witness attribute), 53  
 Account (class in beem.account), 13  
 AccountDoesNotExistException, 34  
 AccountExistsException, 34  
 AccountId() (in module beembase.objects), 55  
 accountopenorders() (beem.market.Market method), 36  
 AccountOptions (class in beembase.objects), 55  
 AccountVotes (class in beem.vote), 49  
 ActiveVotes (class in beem.vote), 49  
 add() (beem.storage.Key method), 45  
 addPrivateKey() (beem.wallet.Wallet method), 51  
 Address (class in beemgraphenebase.account), 60  
 addSigningInformation()  
 (beem.transactionbuilder.TransactionBuilder  
 method), 47  
 AESCipher (class in beem.aes), 19  
 allow() (beem.account.Account method), 14  
 amount (beem.amount.Amount attribute), 20  
 Amount (class in beem.amount), 19  
 Amount (class in beembase.objects), 55  
 appauthor (beem.storage.DataDir attribute), 45

appendMissingSignatures()  
 (beem.transactionbuilder.TransactionBuilder  
 method), 47  
 appendOps() (beem.transactionbuilder.TransactionBuilder  
 method), 47  
 appendSigner() (beem.transactionbuilder.TransactionBuilder  
 method), 47  
 appendWif() (beem.transactionbuilder.TransactionBuilder  
 method), 47  
 appname (beem.storage.DataDir attribute), 45  
 approvewitness() (beem.account.Account method), 14  
 as\_base() (beem.price.Price method), 44  
 as\_quote() (beem.price.Price method), 44  
 asset (beem.amount.Amount attribute), 20  
 asset (beem.asset.Asset attribute), 21  
 Asset (class in beem.asset), 20  
 AssetDoesNotExistException, 34  
 AssetId() (in module beembase.objects), 55  
 assets\_from\_string() (in module beem.utils), 48  
 author (beem.comment.Comment attribute), 30  
 authorperm (beem.comment.Comment attribute), 30  
 available\_balances (beem.account.Account attribute), 14  
 awaitTxConfirmation() (beem.blockchain.Blockchain  
 method), 27

## B

b58decode() (in module beemgraphenebase.base58), 63  
 b58encode() (in module beemgraphenebase.base58), 63  
 balances (beem.account.Account attribute), 14  
 Base58 (class in beemgraphenebase.base58), 62  
 base58CheckDecode() (in module beem-  
 graphenebase.base58), 63  
 base58CheckEncode() (in module beem-  
 graphenebase.base58), 63  
 base58decode() (in module beemgraphenebase.base58),  
 63  
 base58encode() (in module beemgraphenebase.base58),  
 63  
 beem (module), 54  
 beem.account (module), 13

beem.aes (module), 19  
beem.amount (module), 19  
beem.asset (module), 20  
beem.block (module), 26  
beem.blockchain (module), 27  
beem.comment (module), 29  
beem.discussions (module), 32  
beem.exceptions (module), 34  
beem.market (module), 36  
beem.memo (module), 39  
beem.message (module), 41  
beem.notify (module), 42  
beem.price (module), 42  
beem.steem (module), 21  
beem.storage (module), 44  
beem.transactionbuilder (module), 47  
beem.utils (module), 48  
beem.vote (module), 49  
beem.wallet (module), 50  
beem.witness (module), 53  
beemapi (module), 60  
beemapi.exceptions (module), 58  
beembase (module), 57  
beembase.chains (module), 54  
beembase.memo (module), 54  
beembase.objects (module), 55  
beembase.objecttypes (module), 57  
beembase.operationids (module), 57  
beembase.transactions (module), 57  
beemgrapheneapi (module), 66  
beemgraphenebase (module), 65  
beemgraphenebase.account (module), 60  
beemgraphenebase.base58 (module), 62  
beemgraphenebase.bip38 (module), 63  
beemgraphenebase.objects (module), 64  
beemgraphenebase.objecttypes (module), 64  
beemgraphenebase.operationids (module), 64  
beemgraphenebase.transactions (module), 65  
Beneficiaries (class in beembase.objects), 56  
Beneficiary (class in beembase.objects), 56  
Block (class in beem.block), 26  
block\_num (beem.block.Block attribute), 26  
block\_num (beem.block.BlockHeader attribute), 27  
block\_time() (beem.blockchain.Blockchain method), 28  
block\_timestamp() (beem.blockchain.Blockchain method), 28  
Blockchain (class in beem.blockchain), 27  
BlockDoesNotExistException, 34  
BlockHeader (class in beem.block), 27  
blocks() (beem.blockchain.Blockchain method), 28  
body (beem.comment.Comment attribute), 30  
BrainKey (class in beemgraphenebase.account), 61  
broadcast() (beem.steem.Steem method), 22

broadcast() (beem.transactionbuilder.TransactionBuilder method), 47  
buy() (beem.market.Market method), 36

## C

cancel() (beem.market.Market method), 37  
cancel\_subscriptions() (beemapi.websocket.SteemWebsocket method), 59  
cancel\_transfer\_from\_savings() (beem.account.Account method), 14  
category (beem.comment.Comment attribute), 30  
chain\_params (beem.steem.Steem attribute), 22  
changePassphrase() (beem.wallet.Wallet method), 51  
changePassword() (beem.storage.MasterPassword method), 46  
checkBackup() (beem.storage.Configuration method), 10, 44  
claim\_reward\_balance() (beem.account.Account method), 14  
clean\_data() (beem.storage.DataDir method), 45  
clear() (beem.steem.Steem method), 22  
clear() (beem.transactionbuilder.TransactionBuilder method), 47  
clear\_local\_keys() (beem.wallet.Wallet method), 51  
close() (beem.notify.Notify method), 42  
close() (beemapi.websocket.SteemWebsocket method), 59  
Comment (class in beem.comment), 29  
Comment\_discussions\_by\_payout (class in beem.discussions), 32  
comment\_options() (beem.comment.Comment method), 30  
CommentOptionExtensions (class in beembase.objects), 56  
compressed() (beemgraphenebase.account.PublicKey method), 62  
compressedpubkey() (beemgraphenebase.account.PrivateKey method), 62  
config\_defaults (beem.storage.Configuration attribute), 44  
config\_key (beem.storage.MasterPassword attribute), 46  
configStorage (beem.wallet.Wallet attribute), 51  
Configuration (class in beem.storage), 10, 44  
connect() (beem.steem.Steem method), 22  
construct\_authorperm() (in module beem.utils), 48  
construct\_authorpermvoter() (in module beem.utils), 48  
constructTx() (beem.transactionbuilder.TransactionBuilder method), 47  
ContentDoesNotExistException, 34  
convert() (beem.account.Account method), 14  
copy() (beem.amount.Amount method), 20  
copy() (beem.price.Price method), 44  
create() (beem.wallet.Wallet method), 51

create\_account() (beem.steem.Steem method), 22  
 create\_table() (beem.storage.Configuration method), 10, 44  
 create\_table() (beem.storage.Key method), 46  
 created() (beem.wallet.Wallet method), 51  
 curation\_stats() (beem.account.Account method), 14  
 custom\_json() (beem.steem.Steem method), 23

## D

data\_dir (beem.storage.DataDir attribute), 45  
 DataDir (class in beem.storage), 45  
 decode\_memo() (in module beembase.memo), 54  
 decode\_memo\_bts() (in module beembase.memo), 54  
 decodeRPCErrorMsg() (in module beemapi.exceptions), 58  
 decrypt() (beem.aes.AESCipher method), 19  
 decrypt() (beem.memo.Memo method), 41  
 decrypt() (in module beemgraphenebase.bip38), 63  
 decrypt\_wif() (beem.wallet.Wallet method), 51  
 decrypted\_master (beem.storage.MasterPassword attribute), 46  
 decryptEncryptedMaster()  
     (beem.storage.MasterPassword method), 46  
 delegate\_vesting\_shares() (beem.account.Account method), 15  
 delete() (beem.comment.Comment method), 30  
 delete() (beem.storage.Configuration method), 11, 44  
 delete() (beem.storage.Key method), 46  
 derive256address\_with\_version() (beem-graphenebase.account.Address method), 60  
 derive\_permalink() (in module beem.utils), 48  
 deriveChecksum() (beem.storage.MasterPassword method), 46  
 derivesha256address() (beem-graphenebase.account.Address method), 60  
 derivesha512address() (beem-graphenebase.account.Address method), 60  
 disallow() (beem.account.Account method), 15  
 disapprovewitness() (beem.account.Account method), 15  
 Discussions\_by\_active (class in beem.discussions), 32  
 Discussions\_by\_blog (class in beem.discussions), 33  
 Discussions\_by\_cashout (class in beem.discussions), 33  
 Discussions\_by\_children (class in beem.discussions), 33  
 Discussions\_by\_comments (class in beem.discussions), 33  
 Discussions\_by\_created (class in beem.discussions), 33  
 Discussions\_by\_feed (class in beem.discussions), 33  
 Discussions\_by\_hot (class in beem.discussions), 33  
 Discussions\_by\_promoted (class in beem.discussions), 33

Discussions\_by\_trending (class in beem.discussions), 34  
 Discussions\_by\_votes (class in beem.discussions), 34  
 doublesha256() (in module beemgraphenebase.base58), 63  
 downvote() (beem.comment.Comment method), 30

## E

edit() (beem.comment.Comment method), 30  
 encode\_memo() (in module beembase.memo), 54  
 encode\_memo\_bts() (in module beembase.memo), 55  
 encrypt() (beem.aes.AESCipher method), 19  
 encrypt() (beem.memo.Memo method), 41  
 encrypt() (in module beemgraphenebase.bip38), 64  
 encrypt\_wif() (beem.wallet.Wallet method), 51  
 ensure\_full() (beem.account.Account method), 15  
 ExchangeRate (class in beembase.objects), 56  
 exists\_table() (beem.storage.Configuration method), 11, 45  
 exists\_table() (beem.storage.Key method), 46  
 Extension (class in beembase.objects), 56

## F

feed\_publish() (beem.witness.Witness method), 53  
 FilledOrder (class in beem.price), 42  
 finalizeOp() (beem.steem.Steem method), 24  
 follow() (beem.account.Account method), 15  
 formatTime() (in module beem.utils), 48  
 formatTimedelta() (in module beem.utils), 48  
 formatTimeFromNow() (in module beem.utils), 48  
 formatTimeFromNow() (in module beem-graphenebase.transactions), 65  
 formatTimeString() (in module beem.utils), 48

## G

get() (beem.storage.Configuration method), 11, 45  
 get\_account\_history() (beem.account.Account method), 15  
 get\_account\_votes() (beem.account.Account method), 16  
 get\_all\_accounts() (beem.blockchain.Blockchain method), 28  
 get\_balance() (beem.account.Account method), 16  
 get\_balances() (beem.account.Account method), 16  
 get\_bandwidth() (beem.account.Account method), 16  
 get\_block\_interval() (beem.steem.Steem method), 24  
 get\_blog() (beem.account.Account method), 16  
 get\_blog\_account() (beem.account.Account method), 16  
 get\_blog\_entries() (beem.account.Account method), 16  
 get\_brainkey() (beemgraphenebase.account.BrainKey method), 61  
 get\_chain\_properties() (beem.steem.Steem method), 24  
 get\_config() (beem.steem.Steem method), 24  
 get\_conversion\_requests() (beem.account.Account method), 16

`get_curation_reward()` (beem.account.Account method), 16

`get_current_block()` (beem.blockchain.Blockchain method), 28

`get_current_block_num()` (beem.blockchain.Blockchain method), 28

`get_current_median_history()` (beem.steem.Steem method), 24

`get_dynamic_global_properties()` (beem.steem.Steem method), 24

`get_estimated_block_num()` (beem.blockchain.Blockchain method), 28

`get_feed()` (beem.account.Account method), 16

`get_feed_history()` (beem.steem.Steem method), 24

`get_follow_count()` (beem.account.Account method), 16

`get_followers()` (beem.account.Account method), 16

`get_following()` (beem.account.Account method), 16

`get_hardfork_properties()` (beem.steem.Steem method), 24

`get_median_price()` (beem.steem.Steem method), 24

`get_network()` (beem.steem.Steem method), 24

`get_owner_history()` (beem.account.Account method), 16

`get_parent()` (beem.transactionbuilder.TransactionBuilder method), 48

`get_private()` (beemgraphenebase.account.BrainKey method), 61

`get_private()` (beemgraphenebase.account.PasswordKey method), 61

`get_private_key()` (beemgraphenebase.account.BrainKey method), 61

`get_private_key()` (beemgraphenebase.account.PasswordKey method), 61

`get_public()` (beemgraphenebase.account.BrainKey method), 61

`get_public()` (beemgraphenebase.account.PasswordKey method), 61

`get_public_key()` (beemgraphenebase.account.BrainKey method), 61

`get_public_key()` (beemgraphenebase.account.PasswordKey method), 61

`get_reblogged_by()` (beem.comment.Comment method), 30

`get_recharge_time()` (beem.account.Account method), 16

`get_recharge_time_str()` (beem.account.Account method), 16

`get_recharge_timedelta()` (beem.account.Account method), 16

`get_recovery_request()` (beem.account.Account method), 16

`get_reputation()` (beem.account.Account method), 16

`get_request_id()` (beemapi.websocket.SteemWebsocket method), 59

`get_request_id()` (beemgraphenebase.neapi.graphenerpc.GrapheneRPC method), 66

`get_reserve_ratio()` (beem.steem.Steem method), 25

`get_reward_funds()` (beem.steem.Steem method), 25

`get_sbd_per_rshares()` (beem.steem.Steem method), 25

`get_shared_secret()` (in module beembase.memo), 55

`get_steem_per_mvest()` (beem.steem.Steem method), 25

`get_steem_power()` (beem.account.Account method), 16

`get_string()` (beem.market.Market method), 37

`get_vote()` (beem.account.Account method), 16

`get_votes()` (beem.comment.Comment method), 30

`get_voting_power()` (beem.account.Account method), 16

`get_voting_value_SBD()` (beem.account.Account method), 17

`get_withdraw_routes()` (beem.account.Account method), 17

`get_witness_schedule()` (beem.steem.Steem method), 25

`getAccount()` (beem.wallet.Wallet method), 51

`getAccountFromPrivateKey()` (beem.wallet.Wallet method), 51

`getAccountFromPublicKey()` (beem.wallet.Wallet method), 51

`getAccounts()` (beem.wallet.Wallet method), 51

`getAccountsFromPublicKey()` (beem.wallet.Wallet method), 51

`getActiveKeyForAccount()` (beem.wallet.Wallet method), 51

`getAllAccounts()` (beem.wallet.Wallet method), 51

`getBlockParams()` (in module beemgraphenebase.transactions), 65

`getEncryptedMaster()` (beem.storage.MasterPassword method), 46

`getKeyType()` (beem.wallet.Wallet method), 51

`getMemoKeyForAccount()` (beem.wallet.Wallet method), 52

`getOperationNameForId()` (beembase.objects.Operation method), 56

`getOperationNameForId()` (beemgraphenebase.objects.Operation method), 64

`getOperationNameForId()` (in module beembase.operationids), 57

`getOwnerKeyForAccount()` (beem.wallet.Wallet method), 52

`getPostingKeyForAccount()` (beem.wallet.Wallet method), 52

`getPrivateKeyForPublicKey()` (beem.storage.Key method), 46

`getPrivateKeyForPublicKey()` (beem.wallet.Wallet method), 52

`getPublicKeys()` (beem.storage.Key method), 46

`getPublicKeys()` (beem.wallet.Wallet method), 52

`getSimilarAccountNames()` (beem.account.Account



method), 15  
 gphBase58CheckDecode() (in module beem-graphenebase.base58), 63  
 gphBase58CheckEncode() (in module beem-graphenebase.base58), 63  
 GrapheneObject (class in beemgraphenebase.objects), 64  
 GrapheneRPC (class in beemgrapheneapi.graphenerpc), 65

## H

has\_voted() (beem.account.Account method), 17  
 hash\_op() (beem.blockchain.Blockchain static method), 28  
 history() (beem.account.Account method), 17  
 history\_reverse() (beem.account.Account method), 17

## I

id (beem.comment.Comment attribute), 30  
 info() (beem.steem.Steem method), 25  
 init\_aes() (in module beembase.memo), 55  
 init\_aes\_bts() (in module beembase.memo), 55  
 InsufficientAuthorityError, 34  
 interest() (beem.account.Account method), 17  
 InvalidAssetException, 35  
 InvalidMessageSignature, 35  
 InvalidWifiError, 35  
 invert() (beem.price.Price method), 44  
 is\_appbase\_ready() (beemgrapheneapi.graphenerpc.GrapheneRPC method), 66  
 is\_comment() (beem.comment.Comment method), 30  
 is\_empty() (beem.transactionbuilder.TransactionBuilder method), 48  
 is\_fully\_loaded (beem.account.Account attribute), 17  
 is\_irreversible\_mode() (beem.blockchain.Blockchain method), 28  
 is\_main\_post() (beem.comment.Comment method), 30  
 isArgsThisClass() (in module beem-graphenebase.objects), 64  
 items() (beem.storage.Configuration method), 45

## J

json() (beem.amount.Amount method), 20  
 json() (beem.comment.Comment method), 30  
 json() (beem.price.FilledOrder method), 43  
 json() (beem.price.Price method), 44  
 json() (beem.transactionbuilder.TransactionBuilder method), 48  
 json() (beem.vote.Vote method), 49  
 json() (beembase.objects.Operation method), 56  
 json() (beemgraphenebase.objects.GrapheneObject method), 64  
 json\_metadata (beem.comment.Comment attribute), 30

## K

Key (class in beem.storage), 45  
 keyMap (beem.wallet.Wallet attribute), 52  
 KeyNotFound, 35  
 keys (beem.wallet.Wallet attribute), 52  
 keyStorage (beem.wallet.Wallet attribute), 52

## L

list\_operations() (beem.transactionbuilder.TransactionBuilder method), 48  
 listen() (beem.notify.Notify method), 42  
 ListWitnesses (class in beem.witness), 53  
 lock() (beem.wallet.Wallet method), 52  
 locked() (beem.wallet.Wallet method), 52  
 log (in module beemgraphenebase.base58), 63

## M

make\_patch() (in module beem.utils), 49  
 market (beem.price.Price attribute), 44  
 Market (class in beem.market), 36  
 market\_history() (beem.market.Market method), 37  
 market\_history\_buckets() (beem.market.Market method), 37  
 MasterPassword (beem.wallet.Wallet attribute), 51  
 masterpassword (beem.wallet.Wallet attribute), 52  
 MasterPassword (class in beem.storage), 46  
 Memo (class in beem.memo), 39  
 Memo (class in beembase.objects), 56  
 Message (class in beem.message), 41  
 MissingKeyError, 35  
 MissingRequiredActiveAuthority, 58  
 mkdir\_p() (beem.storage.DataDir method), 45

## N

name (beem.account.Account attribute), 17  
 new\_tx() (beem.steem.Steem method), 25  
 newMaster() (beem.storage.MasterPassword method), 46  
 newWallet() (beem.steem.Steem method), 25  
 newWallet() (beem.wallet.Wallet method), 52  
 next\_sequence() (beemgraphenebase.account.BrainKey method), 61  
 NoAccessApi, 58  
 NoApiWithName, 58  
 nodes (beem.storage.Configuration attribute), 11, 45  
 NoMethodWithName, 58  
 normalize() (beemgraphenebase.account.BrainKey method), 61  
 Notify (class in beem.notify), 42  
 NoWalletException, 35

## O

object\_type (in module beembase.objecttypes), 57

object\_type (in module beemgraphenebase.objecttypes), 64  
ObjectId (class in beembase.objects), 56  
ObjectNotInProposalBuffer, 35  
on\_close() (beemapi.websocket.SteemWebsocket method), 59  
on\_error() (beemapi.websocket.SteemWebsocket method), 59  
on\_message() (beemapi.websocket.SteemWebsocket method), 59  
on\_open() (beemapi.websocket.SteemWebsocket method), 59  
Operation (class in beembase.objects), 56  
Operation (class in beemgraphenebase.objects), 64  
operations (in module beemgraphenebase.operationids), 64  
operations() (beembase.objects.Operation method), 56  
operations() (beemgraphenebase.objects.Operation method), 64  
ops (in module beembase.operationids), 57  
ops() (beem.block.Block method), 26  
ops() (beem.blockchain.Blockchain method), 29  
ops\_statistics() (beem.block.Block method), 26  
ops\_statistics() (beem.blockchain.Blockchain method), 29  
Order (class in beem.price), 43  
orderbook() (beem.market.Market method), 37

## P

parent\_author (beem.comment.Comment attribute), 30  
parent\_permlink (beem.comment.Comment attribute), 30  
parse\_time() (in module beem.utils), 49  
password (beem.storage.MasterPassword attribute), 46  
PasswordKey (class in beemgraphenebase.account), 61  
percent (beem.vote.Vote attribute), 50  
Permission (class in beembase.objects), 56  
permlink (beem.comment.Comment attribute), 31  
point() (beemgraphenebase.account.PublicKey method), 62  
post() (beem.comment.Comment method), 31  
Post\_discussions\_by\_payout (class in beem.discussions), 34  
precision (beem.asset.Asset attribute), 21  
prefix (beem.steem.Steem attribute), 25  
Price (class in beem.price), 43  
Price (class in beembase.objects), 56  
print\_info() (beem.account.Account method), 17  
printAsTable() (beem.vote.VotesObject method), 50  
printAsTable() (beem.witness.WitnessesObject method), 53  
PrivateKey (class in beemgraphenebase.account), 61  
process\_block() (beem.notify.Notify method), 42  
process\_block() (beemapi.websocket.SteemWebsocket method), 59

profile (beem.account.Account attribute), 17  
PublicKey (class in beemgraphenebase.account), 62

## Q

Query (class in beem.discussions), 34

## R

recent\_trades() (beem.market.Market method), 38  
RecentByPath (class in beem.comment), 32  
RecentReplies (class in beem.comment), 32  
refresh() (beem.account.Account method), 17  
refresh() (beem.asset.Asset method), 21  
refresh() (beem.block.Block method), 26  
refresh() (beem.block.BlockHeader method), 27  
refresh() (beem.comment.Comment method), 31  
refresh() (beem.vote.Vote method), 50  
refresh() (beem.witness.Witness method), 53  
refresh\_data() (beem.steem.Steem method), 25  
refreshBackup() (beem.storage.DataDir method), 45  
register\_api() (beem.steem.Steem method), 25  
register\_api() (beemgrapheneapi.graphenerpc.GrapheneRPC method), 66  
remove\_from\_dict() (in module beem.utils), 49  
removeAccount() (beem.wallet.Wallet method), 52  
removePrivateKeyFromPublicKey() (beem.wallet.Wallet method), 52  
rep (beem.account.Account attribute), 18  
rep (beem.vote.Vote attribute), 50  
reply() (beem.comment.Comment method), 31  
reputation (beem.vote.Vote attribute), 50  
reset\_subscriptions() (beem.notify.Notify method), 42  
reset\_subscriptions() (beemapi.websocket.SteemWebsocket method), 60  
resolve\_authorperm() (in module beem.utils), 49  
resolve\_authorpermvoter() (in module beem.utils), 49  
resolve\_root\_identifier() (in module beem.utils), 49  
resteem() (beem.comment.Comment method), 32  
reward\_balances (beem.account.Account attribute), 18  
ripemd160() (in module beemgraphenebase.base58), 63  
rpc (beem.wallet.Wallet attribute), 52  
rpcclose() (beemgrapheneapi.graphenerpc.GrapheneRPC method), 66  
rpcconnect() (beemgrapheneapi.graphenerpc.GrapheneRPC method), 66  
RPCConnectionRequired, 35  
rpcexec() (beemapi.steemnode.rpc.SteemNodeRPC method), 57  
rpcexec() (beemapi.websocket.SteemWebsocket method), 60  
rpcexec() (beemgrapheneapi.graphenerpc.GrapheneRPC method), 66

rpclogin() (beemgrapheneapi.graphenerpc.GrapheneRPC method), 66  
 rshares (beem.vote.Vote attribute), 50  
 rshares\_to\_sbd() (beem.steem.Steem method), 25  
 run\_forever() (beemapi.websocket.SteemWebsocket method), 60

## S

SaltException, 63  
 sanitize\_permalink() (in module beem.utils), 49  
 saveEncryptedMaster() (beem.storage.MasterPassword method), 47  
 saving\_balances (beem.account.Account attribute), 18  
 sbd (beem.vote.Vote attribute), 50  
 sell() (beem.market.Market method), 38  
 set\_default\_account() (beem.steem.Steem method), 25  
 set\_expiration() (beem.transactionbuilder.TransactionBuilder method), 48  
 setKeys() (beem.wallet.Wallet method), 52  
 sign() (beem.message.Message method), 41  
 sign() (beem.steem.Steem method), 25  
 sign() (beem.transactionbuilder.TransactionBuilder method), 48  
 sp (beem.account.Account attribute), 18  
 sp\_to\_rshares() (beem.steem.Steem method), 26  
 sp\_to\_sbd() (beem.steem.Steem method), 26  
 sp\_to\_vests() (beem.steem.Steem method), 26  
 sqlDataBaseFile (beem.storage.DataDir attribute), 45  
 sqlite3\_backup() (beem.storage.DataDir method), 45  
 steem (beem.wallet.Wallet attribute), 52  
 Steem (class in beem.steem), 21  
 SteemNodeRPC (class in beemapi.steemnoderpc), 57  
 SteemWebsocket (class in beemapi.websocket), 58  
 stop() (beemapi.websocket.SteemWebsocket method), 60  
 storageDatabase (beem.storage.DataDir attribute), 45  
 str\_to\_bytes() (beem.aes.AESCipher static method), 19  
 stream() (beem.blockchain.Blockchain method), 29  
 suggest() (beemgraphenebase.account.BrainKey method), 61  
 symbol (beem.amount.Amount attribute), 20  
 symbol (beem.asset.Asset attribute), 21  
 symbols() (beem.price.Price method), 44

## T

test\_proposal\_in\_buffer() (in module beem.utils), 49  
 ticker() (beem.market.Market method), 38  
 time (beem.vote.Vote attribute), 50  
 time() (beem.block.Block method), 27  
 time() (beem.block.BlockHeader method), 27  
 title (beem.comment.Comment attribute), 32  
 toJson() (beemgraphenebase.objects.GrapheneObject method), 64  
 total\_balances (beem.account.Account attribute), 18  
 trades() (beem.market.Market method), 39

TransactionBuilder (class in beem.transactionbuilder), 47  
 transfer() (beem.account.Account method), 18  
 transfer\_from\_savings() (beem.account.Account method), 18  
 transfer\_to\_savings() (beem.account.Account method), 18  
 transfer\_to\_vesting() (beem.account.Account method), 18  
 tryUnlockFromEnv() (beem.wallet.Wallet method), 52  
 tuple() (beem.amount.Amount method), 20  
 tx() (beem.steem.Steem method), 26  
 txbuffer (beem.steem.Steem attribute), 26  
 type\_id (beem.account.Account attribute), 18  
 type\_id (beem.asset.Asset attribute), 21  
 type\_id (beem.comment.Comment attribute), 32  
 type\_id (beem.vote.Vote attribute), 50  
 type\_id (beem.witness.Witness attribute), 53

## U

unCompressed() (beemgraphenebase.account.PublicKey method), 62  
 unfollow() (beem.account.Account method), 18  
 UnhandledRPCError, 58  
 unlock() (beem.steem.Steem method), 26  
 unlock() (beem.wallet.Wallet method), 52  
 unlock\_wallet() (beem.memo.Memo method), 41  
 unlocked() (beem.wallet.Wallet method), 52  
 update() (beem.witness.Witness method), 53  
 update\_account\_profile() (beem.account.Account method), 19  
 update\_memo\_key() (beem.account.Account method), 19  
 updateWif() (beem.storage.Key method), 46  
 upvote() (beem.comment.Comment method), 32

## V

verify() (beem.message.Message method), 42  
 verify\_account\_authority() (beem.account.Account method), 19  
 verify\_authority() (beem.transactionbuilder.TransactionBuilder method), 48  
 VestingBalanceDoesNotExistsException, 35  
 vests\_to\_sp() (beem.steem.Steem method), 26  
 virtual\_op\_count() (beem.account.Account method), 19  
 volume24h() (beem.market.Market method), 39  
 Vote (class in beem.vote), 49  
 vote() (beem.comment.Comment method), 32  
 VoteDoesNotExistsException, 35  
 voter (beem.vote.Vote attribute), 50  
 VotesObject (class in beem.vote), 50  
 VotingInvalidOnArchivedPost, 35  
 vp (beem.account.Account attribute), 19

## W

wait\_for\_and\_get\_block() (beem.blockchain.Blockchain

- method), [29](#)
- Wallet (class in beem.wallet), [50](#)
- WalletExists, [35](#)
- WalletLocked, [35](#)
- weight (beem.vote.Vote attribute), [50](#)
- wipe() (beem.storage.Key method), [46](#)
- wipe() (beem.storage.MasterPassword static method), [47](#)
- wipe() (beem.wallet.Wallet method), [52](#)
- withdraw\_vesting() (beem.account.Account method), [19](#)
- Witness (class in beem.witness), [53](#)
- WitnessDoesNotExistsException, [36](#)
- Witnesses (class in beem.witness), [53](#)
- WitnessesObject (class in beem.witness), [53](#)
- WitnessesRankedByVote (class in beem.witness), [53](#)
- WitnessesVotedByAccount (class in beem.witness), [54](#)
- WitnessProps (class in beembase.objects), [56](#)
- WrongMasterPasswordException, [36](#)