beem Documentation

Release 0.1

Holger Nahrstaedt

May 09, 2018

Contents

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.

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 affected by this transaction, e.g.:

- the source account in a transfer
- the accout that buys/sells an asset in the exchange
- the account whos 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")
steem.transfer("<to>", "<amount>", "<asset>", "<memo>", account="<from>")
```

```
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)
```

```python
from beem.market import Market
# Not working at the moment
# market = Market("STEEM:SBD")
# print(market.ticker())
# market.steem.wallet.unlock("wallet-passphrase")
# print(market.sell(300, 100)  # sell 100 STEEM for 300 STEEM/SBD
```

```python
from beem.dex import Dex
# not working at the moment
# dex = Dex()
# dex.steem.wallet.unlock("wallet-passphrase")
```

CHAPTER 3

General

3.1 Installation

3.1.1 Installation

Install with *pip3*:

```
$ sudo apt-get install libffi-dev libssl-dev python-dev
$ pip3 install beem
```

or the with *pip*:

```
$ pip install -U beem
```

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

testnet = Steem(
    "wss://testnet.steem.vc",
    nobroadcast=True,
    bundle=True,
)

testnet.wallet.unlock("supersecret")

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

pprint(testnet.broadcast())
```

3.3.2 Proposing a Transaction

In Steem, you can propose a transactions to any account. This is used to facilitate on-chain multisig transactions. With python-steem, you can do this simply by using the `proposer` attribute:

```
from pprint import pprint
from beem import Steem

testnet = Steem(
    "wss://testnet.steem.vc",
    proposer="test"
)
testnet.wallet.unlock("supersecret")
pprint(testnet.transfer("tst1", 1, "STEEM", account="test"))
```

3.3.3 Simple Sell Script

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

#
# Instanciate Steem (pick network via API node)
```

```
#
steem = Steem(
    "wss://node.testnet.steem.eu",
    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(
    "GOLD:USD",
    steem_instance=steem
)

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

3.3.4 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__":
    #
    # Instanciate Steem (pick network via API node)
    #
    steem = Steem(
        "wss://node.testnet.steem.eu",
        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(
        "GOLD:USD",
        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

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:

CHAPTER 4

beem Libraries

4.1 Steem

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.

4.2 Instances

Default instance to be used when no `steem_instance` is given to the Objects!

```
from beem.instance import shared_steem_instance

account = Account("test")
# is equivalent with
account = Account("test", steem_instance=shared_steem_instance())
```

4.3 Account

Obtaining data of an account.

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

4.4 Amount

For the sake of easier handling of Assets on the blockchain

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

4.5 Asset

4.6 Block

Easily read data in a Block

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

4.7 Blockchain

Read blockchain related data-

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

Read current block and blockchain info

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

Monitor for new blocks ..

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

. . . or each operation individually:

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

4.8 Exceptions

4.9 Market

4.10 Notify

This modules allows yout to be notified of events taking place on the blockchain.

4.11 Price

4.12 Witness

Read data about a witness

```
from beem.witness import Witness
Witness("gtg")
```

CHAPTER 5

Low Level Classes

5.1 Storage

These classes are very low level and are not well documented.

5.1.1 API

5.2 Utilities

5.3 Transaction Builder

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()
```

5.4 Wallet

5.4.1 Create a new wallet

A new wallet can be created by using:

```
from beem import Steem
steem = Steem()
steem.wallet.create("supersecret-passphrase")
```

This will raise an exception if you already have a wallet installed.

5.4.2 Unlocking the wallet for signing

The wallet can be unlocked for signing using

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

5.4.3 Adding a Private Key

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("5xxxxxxxxxxxxxxxxxxxx")
```

Note: The private key has to be either in hexadecimal or in wallet import format (wif) (starting with a `5`).

5.4.4 API

5.5 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()
```

5.5.1 Defintion

5.6 SteemNodeRPC

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

5.6.1 Defintion

5.7 Manual Constructing and Signing of Transactions

Warning: This is a low level class. Do not use this class unless you know what you are doing!

Note: This class is under development and meant for people that are looking into the low level construction and signing of various transactions.

5.7.1 Loading Transactions Class

We load the class for manual transaction construction via:

```
from beembase import transactions, operations
```

5.7.2 Construction

Now we can use the predefined transaction formats, e.g. `Transfer` or `limit_order_create` as follows:

1. define the expiration time
2. define a JSON object that contains all data for that transaction
3. load that data into the corresponding **operations** class
4. collect multiple operations
5. get some blockchain parameters to prevent replay attack
6. Construct the actual **transaction** from the list of operations
7. sign the transaction with the corresponding private key(s)

Example A: Transfer

```
expiration = transactions.formatTimeFromNow(60)
op = operations.Transfer(**{
    "from": "test",
    "to": "test1",
    "amount": "1.000 SBD",
    "memo": ""
})
ops    = [transactions.Operation(op)]
ref_block_num, ref_block_prefix = transactions.getBlockParams(rpc)
```

```
tx      = transactions.Signed_Transaction(ref_block_num=ref_block_num,
                                          ref_block_prefix=ref_block_prefix,
                                          expiration=expiration,
                                          operations=ops)
tx = tx.sign([wif])
```

5.7.3 Broadcasting

For broadcasting, we first need to convert the transactions class into a JSON object. After that, we can broadcast this to the network:

```
# Broadcast JSON to network
rpc.broadcast_transaction(tx.json(), api="network_broadcast"):
```

5.8 Memo

5.8.1 Memo Keys

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 publick 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 retreived with the *get_account* call:

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

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

5.8.2 Memo Message

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.

5.8.3 Example

Encrypting a memo

The high level memo class makes use of the pysteem wallet to obtain keys for the corresponding accounts.

```
from beem.memo import Memo
from beem.account import Account

memoObj = Memo(
    from_account=Account(from_account),
    to_account=Account(to_account)
)
encrypted_memo = memoObj.encrypt(memo)
```

Decoding of a received memo

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

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

# Instantiate Memo for decoding
memo = Memo()

# Unlock wallet
memo.unlock_wallet(getpass())

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

5.8.4 API

CHAPTER 6

Glossary

CHAPTER 7

Indices and tables

- genindex
- modindex
- search