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.

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 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")
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.purge()
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
```

CHAPTER 3

General

3.1 Installation

Warning: install beem will install pycrytodome which is not compatible to pycryto 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

#
# 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 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__":
    #
```

```
    # 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

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

Packages

4.1 beem

4.1.1 beem package

Submodules

beem.account module

beem.aes module

beem.amount module

beem.asset module

beem.steem module

beem.block module

beem.blockchain module

beem.comment module

beem.discussions module

beem.exceptions module

beem.market module

beem.memo module

beem.message module

beem.notify module

beembase.operationids module

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

`beembase.operationids.`**`ops = ['vote', 'comment', 'transfer', 'transfer_to_vesting', 'withdra`**
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', 'withdra`**
Operation ids

beembase.transactions module

Module contents

4.3 beemapi

4.3.1 beemapi package

Submodules

SteemNodeRPC

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

Defintion

`class` `beemapi.steemnoderpc.`**`SteemNodeRPC`**(**args*, ***kwargs*)
This class allows to call API methods exposed by the witness node via websockets.

`__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.graphenewsrpc.RPCError

exception beemapi.exceptions.**NoAccessApi**
Bases: beemgrapheneapi.graphenewsrpc.RPCError

exception beemapi.exceptions.**NoMethodWithName**
Bases: beemgrapheneapi.graphenewsrpc.RPCError

exception beemapi.exceptions.**NumRetriesReached**
Bases: Exception

exception beemapi.exceptions.**UnhandledRPCError**
Bases: beemgrapheneapi.graphenewsrpc.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=''*, **args*, *only_block_id=False*, *on_block=None*, *keep_alive=25*, *num_retries=-1*, ***kwargs*)

Create a websocket connection and request push notifications

Parameters

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

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

- on_block

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

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

Notices:

- `on_block`:

  ```
  '0062f19df70ecf3a478a84b4607d9ad8b3e3b607'
  ```

_SteemWebsocket__set_subscriptions()

__events__ = ['on_block']

__getattr__(*name*)
Map all methods to RPC calls and pass through the arguments

__init__(*urls*, *user=''*, *password=''*, **args*, *only_block_id=False*, *on_block=None*, *keep_alive=25*, *num_retries=-1*, ***kwargs*)

__module__ = 'beemapi.websocket'

_ping()

cancel_subscriptions()

close()
Closes the websocket connection and waits for the ping thread to close

get_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=[]*)

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

Module contents

4.4 beemgraphenebase

4.4.1 beemgraphenebase package

Submodules

beemgraphenebase.account module

beemgraphenebase.base58 module

beemgraphenebase.bip38 module

beemgraphenebase.ecdasig module

beemgraphenebase.objects module

beemgraphenebase.objecttypes module

beemgraphenebase.objecttypes.**object_type = {'base': 1, 'OBJECT_TYPE_COUNT': 3, 'null': 0,**

Object types for object ids

beemgraphenebase.operations module

beemgraphenebase.operationids.**operations = {'demooepration': 0}**

Operation ids

beemgraphenebase.transactions module

Module contents

4.5 beemgrapheneapi

4.5.1 beemgrapheneapi package

Submodules

RPC Interface

Note: This is a low level class that can be used in combination with GrapheneClient

We now need to distinguish functionalities. If we want to only access the blockchain and do not want to perform on-chain operations like transfers or orders, we are fine to interface with any accessible witness node. In contrast, if we want to perform operations that modify the current blockchain state, e.g. construct and broadcast transactions, we are required to interface with a cli_wallet that has the required private keys imported. We here assume:

- port: 8090 - witness
- port: 8092 - wallet

Note: The witness API has a different instruction set than the wallet!

Definition

class beemgrapheneapi.grapheneapi.**GrapheneAPI**(*host*, *port*, *username=''*, *password=''*)

Graphene JSON-HTTP-RPC API

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

Parameters

- **host** (*str*) – Host of the API server
- **port** (*int*) – Port to connect to
- **username** (*str*) – Username for Authentication (if required, defaults to "")
- **password** (*str*) – Password for Authentication (if required, defaults to "")

All RPC commands of the Graphene client are exposed as methods in the class `grapheneapi`. Once an instance of GrapheneAPI is created with host, port, username, and password, e.g.,

```
from grapheneapi import GrapheneAPI
rpc = GrapheneAPI("localhost", 8092, "", "")
```

any call available to that port can be issued using the instance via the syntax rpc.*command*(*parameters*). Example:

```
rpc.info()
```

Note: A distinction has to be made whether the connection is made to a **witness/full node** which handles the blockchain and P2P network, or a **cli-wallet** that handles wallet related actions! The available commands differ drastically!

If you are connected to a wallet, you can simply initiate a transfer with:

```
res = client.transfer("sender","receiver","5", "USD", "memo", True);
```

Again, the witness node does not offer access to construct any transactions, and hence the calls available to the witness-rpc can be seen as read-only for the blockchain.

__getattr__(*name*)

Map all methods to RPC calls and pass through the arguments

rpcexec (*payload*)

Manual execute a command on API (internally used)

param str payload: The payload containing the request return: Servers answer to the query rtype: json raises RPCConnection: if no connction can be made raises UnauthorizedError: if the user is not authorized raise ValueError: if the API returns a non-JSON formated answer

It is not recommended to use this method directly, unless you know what you are doing. All calls available to the API will be wrapped to methods directly:

```
info -> grapheneapi.info()
```

WebsocketRPC

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.graphenewsrpc.**GrapheneWebsocketRPC**(*urls*, *user=''*, *password=''*, ***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 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
- history

Usage:

```
ws = GrapheneWebsocketRPC("ws://10.0.0.16:8090","","")
print(ws.get_account_count())
```

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

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

Module contents

CHAPTER 5

Glossary

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

b

Index

Symbols

B

C

D

G

M

N

O

P

R

S

U