python-steem Documentation

Release 0.1

Fabian Schuh

November 02, 2016

Contents

CHAPTER 1

Python-Steem Libraries

1.1 Installation

1.1.1 Installation

Install with *pip*:

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

Manual installation:

```
$ git clone https://github.com/xeroc/python-steem/
$ cd python-steem
$ python3 setup.py install --user
```

1.1.2 Upgrade

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

1.2 Steem Client

1.2.1 Configuration

class steemapi.steemclient.**ExampleConfig**

The behavior of your program (e.g. reactions on messages) can be defined in a separated class (here called `Config()`. It contains the wallet and witness connection parameters:

The config class is used to define several attributes *and* methods that will be used during API communication. This is particularily useful when dealing with event-driven websocket notifications.

RPC-Only Connections:

The simples setup for this class is a simply RPC:

```
    class Config():
        wallet_host           = "localhost"
        wallet_port           = 8092
```

```
        wallet_user             = ""
        wallet_password         = ""
```

and allows the use of rpc commands similar to the `SteemWalletRPC` class:

```
steem = SteemClient(Config)
print(steem.rpc.info())
print(steem.rpc.get_account("init0"))
print(steem.rpc.get_asset("USD"))
```

All methods within `steem.rpc` are mapped to the corresponding RPC call **of the wallet** and the parameters are handed over directly.

Additional Websocket Connections:

```
class Config():  ## Note the dependency
    wallet_host             = "localhost"
    wallet_port             = 8092
    wallet_user             = ""
    wallet_password         = ""
    witness_url             = "ws://localhost:8090/"
    witness_user            = ""
    witness_password        = ""
```

All methods within `steem.ws` are mapped to the corresponding RPC call of the **full/witness node** and the parameters are handed over directly.

`wallet_host` = 'localhost'
Wallet connection parameters

`witness_url` = 'ws://localhost:8090/'
Witness connection parameter

1.2.2 SteemClient

class `steemapi.steemclient.`**`SteemClient`**(*config*, ***kwargs*)
The `SteemClient` class is an abstraction layer that makes the use of the RPC and the websocket interface easier to use. A part of this abstraction layer is to simplyfy the usage of objects and have an internal objects map updated to reduce unecessary queries (for enabled websocket connections). Advanced developers are of course free to use the underlying API classes instead as well.

Parameters **`config`** (*`class`*) – the configuration class

If a websocket connection is configured, the websocket subsystem can be run by:

```
steem = SteemClient(config)
steem.run()
```

`rpc` = None
RPC connection to the cli-wallet

`ws` = None
Websocket connection to the witness/full node

1.3 SteemStream

This module allows to stream blocks and individual operations from the blockchain and run bots with a minimum of code.

1.3.1 Example

This example code shows all comments starting at block 1893850.

```
from steemapi.steemnoderpc import SteemNodeRPC
from pprint import pprint

rpc = SteemNodeRPC("wss://steemit.com/ws")

for a in rpc.stream("comment", start=1893850):
    pprint(a)
```

1.3.2 Definition

class steemapi.steemnoderpc.**SteemNodeRPC**(*url*, *user=''*, *password=''*, ***kwargs*)

This class allows to call API methods synchronously, without callbacks. It logs in and registers to the APIs:

- database
- history

Parameters

- **url** (*str*) – Websocket URL
- **user** (*str*) – Username for Authentication
- **password** (*str*) – Password for Authentication
- **apis** (*Array*) – List of APIs to register to (default: ["database", "network_broadcast"])

Available APIs

- database
- network_node
- network_broadcast
- history

Usage:

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

stream(*opNames*, **args*, ***kwargs*)

Yield specific operations (e.g. comments) only

Parameters

- **opNames** (*array*) – List of operations to filter for, e.g. vote, comment, transfer, transfer_to_vesting, withdraw_vesting, limit_order_create, limit_order_cancel, feed_publish,

convert, account_create, account_update, witness_update, account_witness_vote, account_witness_proxy, pow, custom, report_over_production, fill_convert_request, comment_reward, curate_reward, liquidity_reward, interest, fill_vesting_withdraw, fill_order,

- **start** (*int*) – Begin at this block

1.4 Asynchronous Steem Client

1.4.1 Configuration

class steemapi.steemasyncclient.**Config**(***kwargs*)

The Config class is used to contain all the parameters needed by SteemAsyncClient in a neat and simple object. The main parameters needed are the witness connection parameters and the mapping of aliases used in code to Steem APIs that are requested to be initialized by SteemAsyncClient on behalf of the code.

The simplest way to create a valid Config object is to pass the keyword arguments to the constructor to initialize the relevant parameters, as the following example demonstrates:

```
config = Config(witness_url      = "ws://localhost:8090/",
                witness_user     = "",
                witness_password = "",
                witness_apis     = {"db": "database",
                                    "broadcast": "network_broadcast"},
                wallet_url       = "ws://localhost:8091/",
                wallet_user      = "",
                wallet_password  = "")
```

But the better way to create the Config object is to put the parameters in YAML configuration file and load it as in the following example:

```
config = Config(config_file = "/path/to/config.yml")
```

Note that you can combine both methods by specifying a config_file but then selectively overriding any of the paramaters from the configuration file by specifying them directly as keyword arguments to the Config constructor.

1.4.2 SteemAsyncClient

class steemapi.steemasyncclient.**SteemAsyncClient**(*config*)

Steem Asynchronous Client

The SteemAsyncClient class is an abstraction layer that makes asynchronous use of the RPC API of either steemd (witness) or cli_wallet (wallet) easy to use.

Parameters **config** (*class*) – the configuration class

Example usage of this class:

```
from steemasyncclient import SteemAsyncClient, Config

@asyncio.coroutine
def print_block_number(steem):
   res = yield from steem.database.get_dynamic_global_properties()
   print(res["head_block_number"])

steem = SteemAsyncClient(Config(witness_url="ws://localhost:8090",
```

```
                                        witness_apis=["database"]))
    steem.run([print_block_number])
```

See more examples of how to use this class in the examples folder.

1.5 Manual Constructing and Signing of Transactions

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

1.5.1 Loading Transactions Class

We load the class for manual transaction construction via:

```
from steembase import transactions
```

1.5.2 Construction

Now we can use the predefined transaction formats, e.g. `vote` or `comment` 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: Vote

```
expiration = transactions.formatTimeFromNow(60)
op = transactions.Vote(
    **{"voter": voter,
       "author": message["author"],
       "permlink": message["permlink"],
       "weight": int(weight)}
)
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])
```

Example A: Comment

```
# Expiration time 60 seconds in the future
expiration = transactions.formatTimeFromNow(60)
op = transactions.Comment(
    **{"parent_author": parent_author,
       "parent_permlink": parent_permlink,
       "author": author,
       "permlink": postPermlink,
       "title": postTitle,
       "body": postBody,
       "json_metadata": ""}
)
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])
```

1.5.3 Broadcasting

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

```
# Convert python class to JSON
tx = transactions.JsonObj(tx)

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

CHAPTER 2

Exchange

2.1 Exchange

2.1.1 Quickstart

```
from pprint import pprint
from steemexchange import SteemExchange

class Config():
    witness_url     = "wss://this.piston.rocks/"
    account         = "xeroc"
    # Either provide a cli-wallet RPC
    wallet_host     = "localhost"
    wallet_port     = 8092
    # or the (active) private key for your account
    wif             = ""

steem = SteemExchange(Config)
pprint(steem.buy(10, "SBD", 100))
pprint(steem.sell(10, "SBD", 100))
pprint(steem.cancel("24432422"))
pprint(steem.returnTicker())
pprint(steem.return24Volume())
pprint(steem.returnOrderBook(2))
pprint(steem.ws.get_order_book(10, api="market_history"))
pprint(steem.returnTradeHistory())
pprint(steem.returnMarketHistoryBuckets())
pprint(steem.returnMarketHistory(300))
pprint(steem.get_lowest_ask())
pprint(steem.get_higest_bid())
pprint(steem.transfer(10, "SBD", "fabian", "foobar"))
```

2.1.2 Definition

class steemexchange.exchange.**SteemExchange**(*config*, ***kwargs*)

This class serves as an abstraction layer for the decentralized exchange within the network and simplifies interaction for trading bots.

Parameters **config** (*config*) – Configuration Class, similar to the example above

This class tries to map the poloniex API around the DEX but has some differences:

- market pairs are denoted as 'quote'_'base', e.g. *USD_BTS*
- 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
- All markets could be considered reversed as well ('BTS_USD')

Usage:

```
from steemexchange import SteemExchange
from pprint import pprint

class Config():
    wallet_host           = "localhost"
    wallet_port           = 8092
    #witness_url           = "ws://localhost:8090/"
    witness_url           = "wss://steemit.com/wstmp2"
    account = "xeroc"

steem = SteemExchange(Config)
pprint(steem.buy(10, "SBD", 100))
pprint(steem.sell(10, "SBD", 100))
pprint(steem.returnTicker())
pprint(steem.return24Volume())
pprint(steem.returnOrderBook(2))
pprint(steem.ws.get_order_book(10, api="market_history"))
pprint(steem.returnTradeHistory())
pprint(steem.returnMarketHistoryBuckets())
pprint(steem.returnMarketHistory(300))
pprint(steem.get_lowest_ask())
pprint(steem.get_higest_bid())
```

buy (*amount*, *quote_symbol*, *rate*, *expiration=604800*, *killfill=False*)

Places a buy order in a given market (buy `quote`, sell `base` in market `quote_base`). If successful, the method will return the order creating (signed) transaction.

Parameters

- **amount** (*number*) – Amount of `quote` to buy
- **quote_symbol** (*str*) – STEEM, or SBD
- **price** (*float*) – price denoted in `base`/`quote`
- **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)

Prices/Rates are denoted in 'base', i.e. the STEEM:SBD market is priced in SBD per STEEM.

Example: in the SBD:STEEM market, a price of 300 means a SBD is worth 300 STEEM

cancel (*orderNumber*)

Cancels an order you have placed in a given market. Requires only the "orderNumber". An order number takes the form `1.7.xxx`.

Parameters **orderNumber** (*str*) – The Order Object ide of the form `1.7.xxxx`

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 formated time for Graphene (*%Y-%m-%dT%H:%M:%S*)

Return type str

getMyAccount()
Returns the structure containing all data relevant to the account specified in the configuration

get_higest_bid()
Return the highest bid.

Example:

```
{'SBD:STEEM': [{'price': 3.08643564387293, 'sbd': 320863, 'steem': 990323}],
 'STEEM:SBD': [{'price': '0.32399833185738391',
                'sbd': 320863,
                'steem': 990323}]}
```

get_lowest_ask()
Return the lowest ask.

Example:

```
{'SBD:STEEM': [{'price': 3.08643564387293, 'sbd': 320863, 'steem': 990323}],
 'STEEM:SBD': [{'price': '0.32399833185738391',
                'sbd': 320863,
                'steem': 990323}]}
```

myAccount = None
The trading account

return24Volume()
Returns the 24-hour volume for all markets, plus totals for primary currencies.

Sample output:

```
{'sbd_volume': 108329.611, 'steem_volume': 355094.043}
```

returnBalances()
Return SBD and STEEM balance of the account

returnMarketHistory(*bucket_seconds=300*, *start_age=3600*, *stop_age=0*)
Return the market history (filled orders).

Parameters

- **bucket_seconds** (*int*) – Bucket size in seconds (see *returnMarketHistoryBuckets()*)
- **start_age** (*int*) – Age (in seconds) of the start of the window (default: 1h/3600)
- **end_age** (*int*) – Age (in seconds) of the end of the window (default: now/0)

Example:

```
{'close_sbd': 2493387,
 'close_steem': 7743431,
 'high_sbd': 1943872,
 'high_steem': 5999610,
 'id': '7.1.5252',
 'low_sbd': 534928,
 'low_steem': 1661266,
 'open': '2016-07-08T11:25:00',
 'open_sbd': 534928,
 'open_steem': 1661266,
 'sbd_volume': 9714435,
```

```
         'seconds': 300,
         'steem_volume': 30088443},
```

returnOpenOrders()
Return open Orders of the account

returnOrderBook(*limit=25*)
Returns the order book for the SBD/STEEM markets in both orientations.

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

Sample output:

```
    {'SBD:STEEM': {'asks': [{'price': 3.086436224481787,
                             'sbd': 318547,
                             'steem': 983175},
                            {'price': 3.086429621198315,
                             'sbd': 2814903,
                             'steem': 8688000}],
                   'bids': [{'price': 3.0864376216446257,
                             'sbd': 545133,
                             'steem': 1682519},
                            {'price': 3.086440512632327,
                             'sbd': 333902,
                             'steem': 1030568}]},
     'STEEM:SBD': {'asks': [{'price': '0.32399827090802763',
                             'sbd': 318547,
                             'steem': 983175},
                            {'price': '0.32399896408839779',
                             'sbd': 2814903,
                             'steem': 8688000}],
                   'bids': [{'price': '0.32399812424109331',
                             'sbd': 545133,
                             'steem': 1682519},
                            {'price': '0.32399782076056660',
                             'sbd': 333902,
                             'steem': 1030568}]}}
```

returnTicker()
Returns the ticker for all markets.

Output Parameters:

- `latest`: Price of the order last filled
- `lowest_ask`: Price of the lowest ask
- `highest_bid`: Price of the highest bid
- `sbd_volume`: Volume of SBD
- `steem_volume`: Volume of STEEM
- `percent_change`: 24h change percentage (in %)

Note: All prices returned by `returnTicker` are in the **reveresed** orientation as the market. I.e. in the SBD:STEEM market, prices are STEEM per SBD. That way you can multiply prices with *1.05* to get a +5%.

Sample Output:

```
        {'SBD:STEEM': {'highest_bid': 3.3222341219615097,
                       'latest': 1000000.0,
                       'lowest_ask': 3.0772668228742615,
                       'percent_change': -0.0,
                       'sbd_volume': 108329611.0,
                       'steem_volume': 355094043.0},
         'STEEM:SBD': {'highest_bid': 0.30100226633322913,
                       'latest': 0.0,
                       'lowest_ask': 0.3249636958897082,
                       'percent_change': 0.0,
                       'sbd_volume': 108329611.0,
                       'steem_volume': 355094043.0}}
```

returnTradeHistory(*time=3600*, *limit=100*)
Returns the trade history for the internal market

Parameters

- **hours** (*int*) – Show the last x seconds of trades (default 1h)
- **limit** (*int*) – amount of trades to show (<100) (default: 100)

sell(*amount*, *quote_symbol*, *rate*, *expiration=604800*, *killfill=False*)
Places a sell order in a given market (sell `quote`, buy `base` in market `quote_base`). If successful, the method will return the order creating (signed) transaction.

Parameters

- **amount** (*number*) – Amount of `quote` to sell
- **quote_symbol** (*str*) – STEEM, or SBD
- **price** (*float*) – price denoted in `base`/`quote`
- **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)

Prices/Rates are denoted in 'base', i.e. the STEEM:SBD market is priced in SBD per STEEM.

Example: in the SBD:STEEM market, a price of 300 means a SBD is worth 300 STEEM

transfer(*amount*, *asset*, *recepient*, *memo=''*)
Transfer SBD or STEEM to another account

Parameters

- **amount** (*float*) – Amount to transfer
- **asset** (*str*) – Asset to transfer ("SBD" or "STEEM")
- **recepient** (*str*) – Recepient of the transfer
- **memo** (*str*) – (Optional) Memo attached to the transfer

CHAPTER 3

Low Level Classes

3.1 SteemWalletRPC

Warning: This is a low level class that can be used in combination with `SteemClient`. Do not use this class unless you know what you are doing!

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!

3.1.1 Definition

class steemapi.steemwalletrpc.**SteemWalletRPC**(**args*, ***kwargs*)

STEEM 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 steem client are exposed as methods in the class `SteemWalletRPC`. Once an instance of SteemWalletRPC is created with host, port, username, and password, e.g.,

```
from steemrpc import SteemRPC
rpc = SteemRPC("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

_confirm(*question*, *default='yes'*)
Confirmation dialog that requires *manual* input.

Parameters

- **question** (*str*) – Question to ask the user
- **default** (*str*) – default answer

Returns Choice of the user

Return type bool

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()
```

3.2 SteemNodeRPC

Warning: This is a low level class that can be used in combination with `SteemClient`. Do not use this class unless you know what you are doing!

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

3.2.1 Defintion

class steemapi.steemnoderpc.**SteemNodeRPC**(*url*, *user=''*, *password=''*, ***kwargs*)
This class allows to call API methods synchronously, without callbacks. It logs in and registers to the APIs:

•database

•history

Parameters

- **url** (*str*) – Websocket URL
- **user** (*str*) – Username for Authentication
- **password** (*str*) – Password for Authentication
- **apis** (*Array*) – List of APIs to register to (default: [”database”, “network_broadcast”])

Available APIs

•database

•network_node

•network_broadcast

•history

Usage:

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

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

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

Symbols

B

C

E

F

G

M

R

S

T

W