python-goloslib Documentation

Release 0.2.3

Fabian Schuh

November 02, 2016

Contents

CHAPTER 1

Python-goloslib Libraries

Python-goloslib libraries extend python-steemlib libraries to GOLOS blockchain.

1.1 Installation

1.1.1 Installation

Install with *pip*:

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

Manual installation:

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

1.1.2 Upgrade

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

1.2 Golos Client

1.2.1 Configuration

1.2.2 GolosClient

1.3 GolosStream

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 golosapi.golosnoderpc import GolosNodeRPC
from pprint import pprint

rpc = GolosNodeRPC("wss://golosit.com/ws")

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

1.3.2 Definition

1.4 Asynchronous Golos Client

1.4.1 Configuration

1.4.2 SteemAsyncClient

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 golosbase 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 golosexchange import GolosExchange

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

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

2.1.2 Definition

CHAPTER 3

Low Level Classes

3.1 GolosWalletRPC

Warning: This is a low level class that can be used in combination with `GolosClient`. 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

3.2 GolosNodeRPC

Warning: This is a low level class that can be used in combination with `GolosClient`. 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