lightsteem Documentation

emre yilmaz

Oct 17, 2018

Contents

1 Features **3**

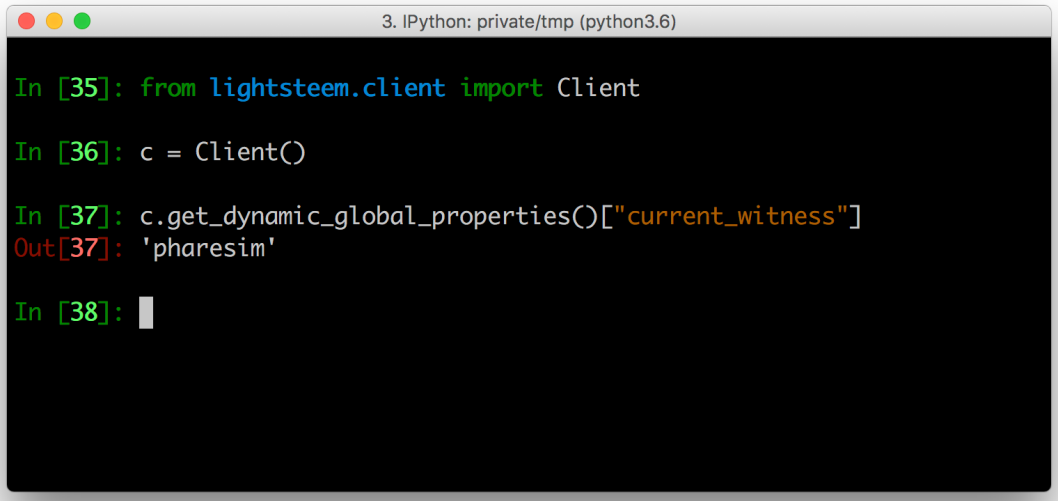
2 Limitations **5**

3 Installation **7**

4 Documentation Pages **9**

4.1 Getting Started . . . 9
4.1.1 Examples . . . 9
4.1.2 Optional parameters of Client . . . 10
4.2 Retry and Failover . . . 10
4.3 Broadcasting Transactions . . . 11
4.3.1 Example: Account Witness Vote . . . 11
4.3.2 Example: Voting for a Post . . . 11
4.3.3 Example: Creating a Post (main Comment) . . . 12
4.3.4 Example: Creating a transfer . . . 12
4.3.5 Example: Bundling Operations . . . 13
4.3.6 Example: Using convert function for SBDs . . . 13
4.4 Batch RPC Calls . . . 14
4.5 Helpers . . . 14
4.6 Account helper . . . 14
4.6.1 Getting account history . . . 14
4.6.2 Getting account followers . . . 15
4.6.3 Getting account followings . . . 16
4.6.4 Getting account ignorers (Muters) . . . 16
4.6.5 Getting account ignorings (Muted list) . . . 16
4.6.6 Getting voting power . . . 16
4.6.7 Getting resource credits . . . 17
4.6.8 Getting account reputation . . . 17
4.7 Amount helper . . . 17
4.8 EventListener Helper . . . 17
4.9 ResourceCredits Helper . . . 18

Lightsteem is a **light** python client to interact with the STEEM blockchain. It's simple and stupid. It doesn't interfere the process between the developer and the STEEM node.



CHAPTER 1

Features

- No hard-coded methods. All potential future appbase methods are automatically supported.
- Retry and Failover support for node errors and timeouts. See *Retry and Failover*.

CHAPTER 2

Limitations

- No support for pre-appbase nodes.
- No magic methods and encapsulation over well-known blockchain objects. (Comment, Post, Account, etc.)

CHAPTER 3

Installation

Lightsteem requires python3.6 and above. Even though it's easy to make it compatible with lower versions, it's doesn't have support by design to keep the library simple.

You can install the library by typing to your console:

```
$ (sudo) pip install lightsteem
```

After that, you can continue with *Getting Started*.

CHAPTER 4

Documentation Pages

4.1 Getting Started

Client class is the primary class you will work with.

```
from lightsteem.client import Client

client = Client()
```

Appbase nodes support different api namespaces.

Client class uses **condenser_api** as default. Follow the official developer portal's api definitions to explore available methods.

4.1.1 Examples

Get Dynamic Global Properties

```
props = client.get_dynamic_global_properties()

print(props)
```

Get Current Reserve Ratio

```
ratio = c('witness_api').get_reserve_ratio()

print(ratio)
```

Get @emrebeyler's account history

```
history = c.get_account_history("emrebeyler", 1000, 10)
```

(continues on next page)

(continued from previous page)

```
for op in history:
    print(op)
```

Get top 100 witness list

```
witness_list = client.get_witnesses_by_vote(None, 100)

print(witness_list)
```

It's the same convention for every api type and every call on appbase nodes.

Important: Since, api_type is set when the client instance is called, it is not thread-safe to share Client instances between threads.

4.1.2 Optional parameters of Client

Even though, you don't need to pass any parameters to the `Client`, you have some options to choose.

`__init__(self, nodes=None, keys=None, connect_timeout=3, read_timeout=30, loglevel=logging.ERROR, chain=None)`

Parameters

- **`nodes`** – A list of appbase nodes. (Defaults: `api.steemit.com`, `appbase.buildteam.io`.)
- **`keys`** – A list of private keys.
- **`connect_timeout`** – Integer. Connect timeout for nodes. (Default:3 seconds.)
- **`read_timeout`** – Integer. Read timeout for nodes. (Default: 30 seconds.)
- **`loglevel`** – Integer. (Ex: logging.DEBUG)
- **`chain`** – String. The blockhain we're working with. (Default: STEEM)

See *Broadcasting Transactions* to find out how to broadcast transactions into the blockchain.

4.2 Retry and Failover

The workflow on retry and failover:

- Send a request to the RPC node
- If the node returns an HTTP status between *400 and 600 or had a timeout, retry the request up to times.
- Sleep time between cycles has an exponential backoff.
- If the node still can't respond, switch tho next available node until exhausting the node list.
- If all nodes are down or giving errors, and the system is out of options, the original exception is raised.

4.3 Broadcasting Transactions

Since Lightsteem supports transaction signing out of the box, you only need to define the operations you want to broadcast.

A typical transaction on STEEM blockchain consists of these fields:

```
{
    "ref_block_num": "..",
    "ref_block_prefix": "..",
    "expiration": "..",
    "operations": [OperationObject, ],
    "extensions": [],
    "signatures": [Signature1,],
}
```

As a library user, you don't need to build the all these information yourself. Since all keys except `operations` can be generated automatically, Lightsteem only asks for a list of operations.

4.3.1 Example: Account Witness Vote

```
from lightsteem.client import Client
from lightsteem.datastructures import Operation

c = Client(
    keys=["<private_key>",])

op = Operation('account_witness_vote', {
        'account': '<your_account>',
        'witness': 'emrebeyler',
        'approve': True,
    })

c.broadcast(op)
```

4.3.2 Example: Voting for a Post

This will vote with a %1. Percent / 100 = Weight. If you want to downvote, use negative weight.

```
from lightsteem.client import Client
from lightsteem.datastructures import Operation

client = Client(
    keys=["<private_key>"]
)

op = Operation('vote', {
    "voter": "emrebeyler",
    "author": "emrebeyler",
    "permlink": "re-hitenkmr-actifit-ios-app-development-contribution-
↪20180816t105311829z",
    "weight": 100,
})
```

(continues on next page)

(continued from previous page)

```
client.broadcast(op)
```

4.3.3 Example: Creating a Post (main Comment)

```
import json

from lightsteem.client import Client
from lightsteem.datastructures import Operation

client = Client(
    keys=["<posting_key>"]
)

post = Operation('comment', {
    "parent_author": None,
    "parent_permlink": "steemit",
    "author": "emrebeyler",
    "permlink": "api-steemit-is-down",
    "title": "api.steemit.com is down",
    "body": "Body of the post",
    "json_metadata": json.dumps({"tags": "steemit steem lightsteem"})
})

resp =client.broadcast(post)

print(resp)
```

Posts are actually Comment objects and same with replies. This example creates a main comment (Post) on the blockchain.

Notes:

- parent_author should be None for posts.
- parent_permlink should be the first tag you use in the post.

If you fill parent_author and parent_permlink with actual post information, you will have a reply. (comment)

4.3.4 Example: Creating a transfer

```
from lightsteem.client import Client
from lightsteem.datastructures import Operation


c = Client(
    keys=["active_key",])

op = Operation('transfer', {
            'from': 'emrebeyler',
            'to': '<receiver_1>',
            'amount': '0.001 SBD',
            'memo': 'test1!'
        })
```

(continues on next page)

(continued from previous page)

```
c.broadcast(ops)
```

4.3.5 Example: Bundling Operations

It's also possible to bundle multiple operations into one transaction:

```
from lightsteem.client import Client
from lightsteem.datastructures import Operation


c = Client(
    keys=["active_key",])

ops = [
    Operation('transfer', {
        'from': 'emrebeyler',
        'to': '<receiver_1>',
        'amount': '0.001 SBD',
        'memo': 'test1!'
    }),
    Operation('transfer', {
        'from': 'emrebeyler',
        'to': '<receiver_2>',
        'amount': '0.001 SBD',
        'memo': 'test2!'
    }),

]

c.broadcast(ops)
```

4.3.6 Example: Using convert function for SBDs

```
from lightsteem.client import Client
from lightsteem.datastructures import Operation

client = Client(
    keys=["<active_key>"]
)
client.broadcast(
    Operation('convert', {
        "owner": "emrebeyler,
        "amount": "0.500 SBD",
        "requestid": 1,
    })
)
```

Note: requestid and the owner is unique together.

Important: Since, lightsteem doesn't introduce any encapsulation on operations, you are responsible to create operation data yourself. To find out the specs for each operation, you may review the block explorers for raw data or

the source code of steemd.

4.4 Batch RPC Calls

Appbase nodes support multiple RPC calls in one HTTP request. (Maximum is 50.). If you want to take advantage of this:

```
from lightsteem.client import Client

c = Client()

c.get_block(24858937, batch=True)
c.get_block(24858938, batch=True)

blocks = c.process_batch()

print(blocks)
```

This will create one request, but you will have two block details.

Important: This feature is not thread-safe. Every instance has a simple queue (list) as their property, and it's flushed every time the `process_batch` is called.

4.5 Helpers

Lightsteem has a target to define helper classes for well known blockchain objects. This is designed in that way to prevent code repeat on client (library user) side.

It's possible to use lightsteem for just a client. However, if you need to get an account's history, or get followers of account, you may use the helpers module.

4.6 Account helper

This class defines an Account in the STEEM blockchain.

```
from lightsteem.client import Client
c = Client()
account = c.get_account('emrebeyler')
```

When you execute that script in your REPL, lightsteem makes a RPC call to get the account data from the blockchain. Once you initialized the Account instance, you have access to these helper methods:

4.6.1 Getting account history

With this method, you can traverse entire history of a STEEM account.

history(self, account=None, limit=1000, filter=None, exclude=None, order="desc", only_operation_data=True,

start_at=None, stop_at=None):

Parameters

- **account** – (string) The username.
- **limit** – (integer) Batch size per call.
- **filter** – (list:string) Operation types to filter
- **exclude** – (list:s tring) Operation types to exclude
- **order** – (string) asc or desc.
- **only_operation_data** – (bool) If false, returns in the raw format. (Includes transaction information.)
- **start_at** – (datetime.datetime) Starts after that time to process ops.
- **stop_at** – (datetime.datetime) Stops at that time while processing ops.

account_history is an important call for the STEEM applications. A few use cases:

- Getting incoming delegations
- Filtering transfers on specific accounts
- Getting author, curation rewards

etc.

Example: Get all incoming STEEM of binance account in the last 7 days

```
import datetime

from lightsteem.client import Client
from lightsteem.helpers.amount import Amount

client = Client()
account = client.account('deepcrypto8')

one_week_ago = datetime.datetime.utcnow() -
    datetime.timedelta(days=7)
total_steem = 0
for op in account.history(
        stop_at=one_week_ago,
        filter=["transfer"]):

    if op["to"] != "deepcrypto8":
        continue

    total_steem += Amount(op["amount"]).amount

print("Total STEEM deposited to Binance", total_steem)
```

4.6.2 Getting account followers

```
from lightsteem.client import Client

client = Client()
account = client.account('deepcrypto8')
```

(continues on next page)

(continued from previous page)

```
print(account.followers())
```

Output will be a list of usernames. (string)

4.6.3 Getting account followings

```
from lightsteem.client import Client

client = Client()
account = client.account('emrebeyler')

print(account.following())
```

Output will be a list of usernames. (string)

4.6.4 Getting account ignorers (Muters)

```
from lightsteem.client import Client

client = Client()
account = client.account('emrebeyler')

print(account.ignorers())
```

4.6.5 Getting account ignorings (Muted list)

```
from lightsteem.client import Client

client = Client()
account = client.account('emrebeyler')

print(account.ignorings())
```

4.6.6 Getting voting power

This helper method determines the account's voting power. In default, It considers account's regenerated VP. (Actual VP)

If you want the VP at the time the last vote casted, you can pass consider_regeneration=False.

```
from lightsteem.client import Client

client = Client()
account = client.account('emrebeyler')

print(account.vp())
print(account.vp(consider_regeneration=False))
```

4.6.7 Getting resource credits

This helper method determines the account's resource credits in percent. In default, It considers account's regenerated RC. (Actual RC)

If you want the Rc at the time the last vote casted, you can pass consider_regeneration=False.

```
from lightsteem.client import Client

client = Client()
account = client.account('emrebeyler')

print(account.rc())
print(account.rc(consider_regeneration=False))
```

4.6.8 Getting account reputation

```
from lightsteem.client import Client

client = Client()
account = client.account('emrebeyler')

print(account.reputation())
```

Default precision is 2. You can set it by passing precision=N parameter.

4.7 Amount helper

A simple class to convert "1234.1234 STEEM" kind of values to Decimal.

```
from lightsteem.helpers.amount import Amount

amount = Amount("42.5466 STEEM")

print(amount.amount)
print(amount.symbol)
```

4.8 EventListener Helper

EventListener is a helper class to listen specific operations (events) on the blockchain.

Stream blockchain for the incoming transfers related to a specific account

```
from lightsteem.helpers.event_listener import EventListener
from lightsteem.client import Client

client = Client()
events = EventListener(client)

for transfer in events.on('transfer', filter_by={"to": "emrebeyler"}):
    print(transfer)
```

Stream for incoming vote actions

```
events = EventListener(client)

for witness_vote in events.on('account_witness_vote', filter_by={"witness":
↪"emrebeyler"}):
    print(witness_vote)
```

Conditions via callables

Stream for the comments and posts tagged with utopian-io.

```
from lightsteem.client import Client
from lightsteem.helpers.event_listener import EventListener

import json

c = Client()
events = EventListener(c)

def filter_tags(comment_body):
    if not comment_body.get("json_metadata"):
        return False

    try:
        tags = json.loads(comment_body["json_metadata"])["tags"]
    except KeyError:
        return False
    return "utopian-io" in tags


for op in events.on("comment", condition=filter_tags):
    print(op)
```

EventListener class also has

- start_block
- end_block

params that you can limit the streaming process into specific blocks.

4.9 ResourceCredits Helper

ResourceCredits class has a simple helper function to get RC costs on specific operations.

For example, if you want to learn about how much resource credit will be exhausted for an **account_claim** operation:

```
from lightsteem.client import Client
from lightsteem.datastructures import Operation

client = Client(keys=[<your_active_key>])

op = Operation(
    'claim_account',
    {
        "creator": "emrebeyler",
        "fee": "0.000 STEEM",
```

(continues on next page)

(continued from previous page)

```
        "extensions": [],
    }
)

print(client.rc().get_cost(op))
```