python-binance-chain Documentation

Release 0.2.0

Sam McHardy

Jun 01, 2021

Contents

This is an unofficial Python3 wrapper for the Binance Chain API. I am in no way affiliated with Binance, use at your own risk.

PyPi https://pypi.python.org/pypi/python-binance-chain

Source code https://github.com/sammchardy/python-binance-chain

CHAPTER 1

Features

- Support for Testnet and Production *environments*, along with user defined environment
- HTTP API *sync* and *async* implementations
- *Async Websockets* with auto-reconnection and backoff retry algorithm
- HTTP RPC Node *sync* and *async* implementations
- Advanced async *Pooled HTTP RPC Node client* spreading requests over available peers
- *Async Node RPC Websockets* with auto-reconnection and backoff retry algorithm
- *Wallet* creation from private key or mnemonic or new wallet with random mnemonic
- Wallet handling account sequence for transactions
- Broadcast Transactions over *HTTP* and *RPC* with helper classes for limit buy and sell
- *Sign transactions* and use the signed message how you want
- *Ledger hardware wallet* device (Ledger Blue, Nano S & Nano X) support for signing messages
- Async *Depth Cache* to keep a copy of the order book locally
- *Signing Service Support* for binance-chain-signing-service
- Support for HTTP and HTTPS proxies and to override Requests and AioHTTP settings
- UltraJson the ultra fast JSON parsing library for efficient message handling
- Strong Python3 typing to reduce errors
- pytest *test suite*
- Response exception handling

Read the Changelog

CHAPTER 2

Recommended Resources

- Binance Chain Forum
- Binance Chain Telegram
- Binance Chain API
- Tendermint Docs
- Get Testnet Funds

CHAPTER 3

Quick Start

```
pip install python-binance-chain
```

If having issues with secp256k1 check the Installation instructions for the sec256k1-py library

If using the production server there is no need to pass the environment variable.

```
from binance_chain.http import HttpApiClient
from binance_chain.constants import KlineInterval
from binance_chain.environment import BinanceEnvironment

# initialise with Testnet environment
testnet_env = BinanceEnvironment.get_testnet_env()
client = HttpApiClient(env=testnet_env)

# Alternatively pass no env to get production
prod_client = HttpApiClient()

# connect client to different URL using custom environments, see below

# get node time
time = client.get_time()

# get node info
node_info = client.get_node_info()

# get validators
validators = client.get_validators()

# get peers
peers = client.get_peers()

# get account
account = client.get_account('tbnb185tqzq3j6y7yep85lncaz9qeectjxqe5054cgn')
```

(continues on next page)

(continued from previous page)

```
# get account sequence
account_seq = client.get_account_sequence('tbnb185tqzq3j6y7yep85lncaz9qeectjxqe5054cgn
↪')

# get markets
markets = client.get_markets()

# get fees
fees = client.get_fees()

# get order book
order_book = client.get_order_book('NNB-0AD_BNB')

# get klines
klines = client.get_klines('NNB-338_BNB', KlineInterval.ONE_DAY)

# get closed orders
closed_orders = client.get_closed_orders('tbnb185tqzq3j6y7yep85lncaz9qeectjxqe5054cgn
↪')

# get open orders
open_orders = client.get_open_orders('tbnb185tqzq3j6y7yep85lncaz9qeectjxqe5054cgn')

# get ticker
ticker = client.get_ticker('NNB-0AD_BNB')

# get trades
trades = client.get_trades(limit=2)

# get order
order = client.get_order('9D0537108883C68B8F43811B780327CE97D8E01D-2')

# get trades
trades = client.get_trades()

# get transactions
transactions = client.get_transactions(address=
↪'tbnb1n5znwyygs0rghr6rsydhsqe8e6ta3cqatucsqp')

# get transaction
transaction = client.get_transaction(
↪'95DD6921370D74D0459590268B439F3DD49F6B1D090121AFE4B2183C040236F3')
```

See API docs for more information.

CHAPTER 4

Async HTTP Client

An implementation of the HTTP Client above using aiohttp instead of requests

Use the async *create* classmethod to initialise an instance of the class.

All methods are otherwise the same as the HttpApiClient

```
from binance_chain.http import AsyncHttpApiClient
from binance_chain.environment import BinanceEnvironment
import asyncio

loop = None

async def main():
    global loop

    env = BinanceEnvironment.get_testnet_env()

    # initialise the class using the classmethod
    client = await AsyncHttpApiClient.create(env)
    wallet = Wallet(private_key=priv_key, env=env)

    print(json.dumps(await client.get_time(), indent=2))

    while True:
        print("doing a sleep")
        await asyncio.sleep(20, loop=loop)


if __name__ == "__main__":

    loop = asyncio.get_event_loop()
    loop.run_until_complete(main())
```

CHAPTER 5

Environments

Binance Chain offers a Production system and Testnet.

If using the Production system there is no need to pass an environment as this is the default.

To create and use the Testnet environment is as easy as

```
from binance_chain.environment import BinanceEnvironment

# initialise with Testnet environment
testnet_env = BinanceEnvironment.get_testnet_env()
```

You may also create your own custom environments, this may be useful such as connecting to a Node RPC client

```
from binance_chain.environment import BinanceEnvironment

# create custom environment
my_env = BinanceEnvironment(api_url="<api_url>", wss_url="<wss_url>", hrp="<hrp>")
```

See API docs for more information.

CHAPTER 6

Wallet

See API docs for more information.

The wallet is required if you want to place orders, transfer funds or freeze and unfreeze tokens.

You may also use the *Ledger Wallet class* to utilise your Ledger Hardware Wallet for signing.

It can be initialised with your private key or your mnemonic phrase.

You can additionally provide BIP39 passphrase and derived wallet id.

Note that the BinanceEnvironment used for the wallet must match that of the HttpApiClient, testnet addresses will not work on the production system.

The Wallet class can also create a new account for you by calling the *Wallet.create_random_wallet()* function, see examples below

Initialise from Private Key

```
from binance_chain.wallet import Wallet
from binance_chain.environment import BinanceEnvironment

testnet_env = BinanceEnvironment.get_testnet_env()
wallet = Wallet('private_key_string', env=testnet_env)
print(wallet.address)
print(wallet.private_key)
print(wallet.public_key_hex)
```

Initialise from Mnemonic

```
from binance_chain.wallet import Wallet
from binance_chain.environment import BinanceEnvironment

testnet_env = BinanceEnvironment.get_testnet_env()
wallet = Wallet.create_wallet_from_mnemonic('mnemonic word string',
                                            passphrase='optional passphrase',
                                            child=0,
                                            env=testnet_env)
```

(continues on next page)

(continued from previous page)

```
print(wallet.address)
print(wallet.private_key)
print(wallet.public_key_hex)
```

Initialise by generating a random Mneomonic

```
from binance_chain.wallet import Wallet
from binance_chain.environment import BinanceEnvironment

testnet_env = BinanceEnvironment.get_testnet_env(, env=testnet_env)
wallet = Wallet.create_random_wallet(env=env)
print(wallet.address)
print(wallet.private_key)
print(wallet.public_key_hex)
```

CHAPTER 7

Broadcast Messages on HttpApiClient

See API docs for more information.

Requires a Wallet to have been created.

The Wallet will increment the request sequence when broadcasting messages through the HttpApiClient.

If the sequence gets out of sync call *wallet.reload_account_sequence(client)*, where client is an instance of HttpApiClient.

Place Order

General case

```
from binance_chain.http import HttpApiClient
from binance_chain.messages import NewOrderMsg
from binance_chain.wallet import Wallet
from binance_chain.constants import TimeInForce, OrderSide, OrderType
from decimal import Decimal

wallet = Wallet('private_key_string')
client = HttpApiClient()

# construct the message
new_order_msg = NewOrderMsg(
    wallet=wallet,
    symbol="ANN-457_BNB",
    time_in_force=TimeInForce.GOOD_TILL_EXPIRE,
    order_type=OrderType.LIMIT,
    side=OrderSide.BUY,
    price=Decimal(0.000396000),
    quantity=Decimal(12)
)
# then broadcast it
res = client.broadcast_msg(new_order_msg, sync=True)
```

Limit Order Buy

```
from binance_chain.messages import LimitOrderBuyMsg

limit_order_msg = LimitOrderBuyMsg(
    wallet=wallet,
    symbol='ANN-457_BNB',
    price=0.000396000,
    quantity=12
)
```

Limit Order Sell

```
from binance_chain.messages import LimitOrderSellMsg

limit_order_msg = LimitOrderSellMsg(
    wallet=wallet,
    symbol='ANN-457_BNB',
    price=0.000396000,
    quantity=12
)
```

Cancel Order

```
from binance_chain.http import HttpApiClient
from binance_chain.messages import CancelOrderMsg
from binance_chain.wallet import Wallet

wallet = Wallet('private_key_string')
client = HttpApiClient()

# construct the message
cancel_order_msg = CancelOrderMsg(
    wallet=wallet,
    order_id="order_id_string",
    symbol='ANN-457_BNB',
)
# then broadcast it
res = client.broadcast_msg(cancel_order_msg, sync=True)
```

Freeze Tokens

```
from binance_chain.http import HttpApiClient
from binance_chain.messages import FreezeMsg
from binance_chain.wallet import Wallet
from decimal import Decimal

wallet = Wallet('private_key_string')
client = HttpApiClient()

# construct the message
freeze_msg = FreezeMsg(
    wallet=wallet,
    symbol='BNB',
    amount=Decimal(10)
)
# then broadcast it
res = client.broadcast_msg(freeze_msg, sync=True)
```

Unfreeze Tokens

```
from binance_chain.http import HttpApiClient
from binance_chain.messages import UnFreezeMsg
from binance_chain.wallet import Wallet
from decimal import Decimal

wallet = Wallet('private_key_string')
client = HttpApiClient()

# construct the message
unfreeze_msg = UnFreezeMsg(
    wallet=wallet,
    symbol='BNB',
    amount=Decimal(10)
)
# then broadcast it
res = client.broadcast_msg(unfreeze_msg, sync=True)
```

Transfer Tokens

```
from binance_chain.http import HttpApiClient
from binance_chain.messages import TransferMsg
from binance_chain.wallet import Wallet

wallet = Wallet('private_key_string')
client = HttpApiClient()

transfer_msg = TransferMsg(
    wallet=wallet,
    symbol='BNB',
    amount=1,
    to_address='<to address>',
    memo='Thanks for the beer'
)
res = client.broadcast_msg(transfer_msg, sync=True)
```

Transfer Multiple Tokens

```
from binance_chain.http import HttpApiClient
from binance_chain.messages import TransferMsg, Transfer
from binance_chain.wallet import Wallet

wallet = Wallet('private_key_string')
client = HttpApiClient()

multi_transfer_msg = TransferMsg(
    wallet=wallet,
    transfers=[
        Transfer(symbol='ETH.B', amount=1),
        Transfer(symbol='BNB', amount=1),
    ],
    to_address='<to address>',
    memo='Thanks for the beer'
)
res = client.broadcast_msg(multi_transfer_msg, sync=True)
```

Vote for proposal

```
from binance_chain.http import HttpApiClient
from binance_chain.messages import VoteMsg
from binance_chain.wallet import Wallet
from binance_chain.constants import VoteOption

wallet = Wallet('private_key_string')
client = HttpApiClient()

vote_msg = VoteMsg(
    wallet=wallet,
    proposal_id=1,
    vote_option=VoteOption.YES
)
res = client.broadcast_msg(vote_msg, sync=True)
```

CHAPTER 8

Sign Transaction

If you want to simply sign a transaction you can do that as well.

This is a transfer example

```
from binance_chain.messages import TransferMsg, Signature
from binance_chain.wallet import Wallet

wallet = Wallet('private_key_string')

transfer_msg = TransferMsg(
    wallet=wallet,
    symbol='BNB',
    amount=1,
    to_address='<to address>'
)
signed_msg = Signature(transfer_msg).sign()
```

CHAPTER 9

Websockets

See API docs for more information.

```
import asyncio

from binance_chain.websockets import BinanceChainSocketManager
from binance_chain.environment import BinanceEnvironment

testnet_env = BinanceEnvironment.get_testnet_env()

address = 'tbnb...'
loop = None

async def main():
    global loop

    async def handle_evt(msg):
        """Function to handle websocket messages
        """
        print(msg)

    # connect to testnet env
    bcsm = await BinanceChainSocketManager.create(loop, handle_evt, address,␣
↪env=testnet_env)

    # subscribe to relevant endpoints
    await bcsm.subscribe_orders(address)
    await bcsm.subscribe_market_depth(["FCT-B60_BNB", "0KI-0AF_BNB"])
    await bcsm.subscribe_market_delta(["FCT-B60_BNB", "0KI-0AF_BNB"])
    await bcsm.subscribe_trades(["FCT-B60_BNB", "0KI-0AF_BNB"])
    await bcsm.subscribe_ticker(["FCT-B60_BNB", "0KI-0AF_BNB"])

    while True:
        print("sleeping to keep loop open")
        await asyncio.sleep(20, loop=loop)
```

(continues on next page)

(continued from previous page)

```
if __name__ == "__main__":

    loop = asyncio.get_event_loop()
    loop.run_until_complete(main())
```

Unsubscribe

```
# with an existing BinanceChainSocketManager instance

await bcsm.unsubscribe_orders()

# can unsubscribe from a particular symbol, after subscribing to multiple
await bcsm.subscribe_market_depth(["0KI-0AF_BNB"])
```

Close Connection

```
# with an existing BinanceChainSocketManager instance

await bcsm.close_connection()
```

CHAPTER 10

Node RPC HTTP

See API docs for more information.

The binance_chain.http.HttpApiClient has a helper function *get_node_peers()* which returns a list of peers with Node RPC functionality

```
from binance_chain.http import HttpApiClient, PeerType
from binance_chain.node_rpc import HttpRpcClient

httpapiclient = HttpApiClient()

# get a peer that support node requests
peers = httpapiclient.get_node_peers()
listen_addr = peers[0]['listen_addr']

# connect to this peer
rpc_client = HttpRpcClient(listen_addr)

# test some endpoints
abci_info = rpc_client.get_abci_info()
consensus_state = rpc_client.dump_consensus_state()
genesis = rpc_client.get_genesis()
net_info = rpc_client.get_net_info()
num_unconfirmed_txs = rpc_client.get_num_unconfirmed_txs()
status = rpc_client.get_status()
health = rpc_client.get_health()
unconfirmed_txs = rpc_client.get_unconfirmed_txs()
validators = rpc_client.get_validators()

block_height = rpc_client.get_block_height(10)
```

CHAPTER 11

Node RPC HTTP Async

An aiohttp implementation of the Node RPC HTTP API.

Use the async *create* classmethod to initialise an instance of the class.

All methods are the same as the binance_chain.node_rpc.http.HttpRpcClient.

```
from binance_chain.node_rpc.http import AsyncHttpRpcClient
from binance_chain.http import AsyncHttpApiClient, PeerType
from binance_chain.environment import BinanceEnvironment
import asyncio

loop = None

async def main():
    global loop

    testnet_env = BinanceEnvironment.get_testnet_env()

    # create the client using the classmethod
    http_client = await AsyncHttpApiClient.create(env=testnet_env)

    peers = await http_client.get_node_peers()
    listen_addr = peers[0]['listen_addr']

    rcp_client = await AsyncHttpRpcClient.create(listen_addr)

    print(json.dumps(await rcp_client.get_abci_info(), indent=2))

    while True:
        print("doing a sleep")
        await asyncio.sleep(20, loop=loop)


if __name__ == "__main__":
```

(continues on next page)

(continued from previous page)

```
    loop = asyncio.get_event_loop()
    loop.run_until_complete(main())
```

CHAPTER 12

Broadcast Messages on Node RPC HTTP Client

Requires a Wallet to have been created

The Wallet will increment the request sequence when broadcasting messages through the HttpApiClient.

If the sequence gets out of sync call *wallet.reload_account_sequence(client)*, where client is an instance of HttpApiClient.

Place Order

```
from binance_chain.node_rpc import HttpRpcClient
from binance_chain.messages import LimitOrderBuyMsg
from binance_chain.wallet import Wallet
from binance_chain.constants import RpcBroadcastRequestType

wallet = Wallet('private_key_string')
rpc_client = HttpRpcClient(listen_addr)

limit_order_msg = LimitOrderBuyMsg(
    wallet=wallet,
    symbol='ANN-457_BNB',
    price=0.000396000,
    quantity=12
)

# then broadcast it, by default in synchronous mode
res = rpc_client.broadcast_msg(limit_order_msg)

# alternative async request
res = rpc_client.broadcast_msg(new_order_msg, request_type=RpcBroadcastRequestType.
↪ASYNC)

# or commit request
res = rpc_client.broadcast_msg(new_order_msg, request_type=RpcBroadcastRequestType.
↪COMMIT)
```

Other messages can be constructed similar to examples above

CHAPTER 13

Pooled Node RPC Client

This client connects to all available peer nodes in the network and spreads requests across them.

This helps reduce API rate limit errors.

The interface is the same as the above HttpRpcClient and AsyncHttpRpcClient classes for consistency.

Requests can be sent using asyncio gather note to check the number of clients connected to and not exceed that amount

```
import asyncio
from binance_chain.node_rpc.pooled_client import PooledRpcClient


async def main():

    # initialise the client, default production environment
    client = await PooledRpcClient.create()

    # optionally include an environment
    testnet_env = BinanceEnvironment.get_testnet_env()
    client = await PooledRpcClient.create(env=testnet_env)

    # show the number of peers connected with the num_peers property
    print(f"Connected to {client.num_peers} peers")

    # requests can be send in bulk using asyncio gather
    for i in range(0, 5):
        res = await asyncio.gather(
            client.get_abci_info(),
            client.get_consensus_state(),
            client.get_net_info(),
            client.get_status(),
            client.get_health(),
        )
        print(f'{i}: {res}')
```

(continues on next page)

(continued from previous page)

```
    print(await client.get_block(1000))

    print(await client.get_blockchain_info(1000, 2000))

if __name__ == "__main__":

    loop = asyncio.get_event_loop()
    loop.run_until_complete(main())
```

To keep the peer connections up to date you may re-initialise the list of peers by calling the *initialise_peers* function

```
client.initialise_peers()
```

CHAPTER 14

Node RPC Websockets

See API docs for more information.

For subscribe query examples see the documentation here

```
import asyncio

from binance_chain.http import HttpApiClient
from binance_chain.environment import BinanceEnvironment
from binance_chain.node_rpc.websockets import WebsocketRpcClient

loop = None

async def main():
    global loop

    async def handle_evt(msg):
        print(msg)

    # find node peers on testnet
    testnet_env = BinanceEnvironment.get_testnet_env()
    client = HttpApiClient(testnet_env)

    peers = client.get_node_peers()

    # construct websocket listen address - may not be correct
    listen_addr = re.sub(r"^https?:\/\/", "tcp://", peers[0]['listen_addr'])

    # create custom environment for RPC Websocket
    node_env = BinanceEnvironment(
        api_url=testnet_env.api_url,
        wss_url=listen_addr,
        hrp=testnet_env.hrp
    )

    wrc = await WebsocketRpcClient.create(loop, handle_evt, env=node_env)
```

(continues on next page)

(continued from previous page)

```
    await wrc.subscribe("tm.event = 'NewBlock'")
    await wrc.abci_info()

    while True:
        print("sleeping to keep loop open")
        await asyncio.sleep(20, loop=loop)


if __name__ == "__main__":

    loop = asyncio.get_event_loop()
    loop.run_until_complete(main())
```

Unsubscribe

```
# with an existing WebsocketRpcClient instance

await wrc.unsubscribe("tm.event = 'NewBlock'")
```

Unsubscribe All

```
# with an existing WebsocketRpcClient instance

await wrc.unsubscribe_all()
```

CHAPTER 15

Depth Cache

Follow the order book for a specified trading pair.

Note: This may not be 100% reliable as the response info available from Binance Chain may not always match up

```
from binance_chain.depthcache import DepthCacheManager
from binance_chain.environment import BinanceEnvironment
from binance_chain.http import HttpApiClient

dcm = None
loop = None


async def main():
    global dcm1, loop

    async def process_depth(depth_cache):
        print("symbol {}".format(depth_cache.symbol))
        print("1: top 5 asks")
        print(depth_cache.get_asks()[:5])
        print("1: top 5 bids")
        print(depth_cache.get_bids()[:5])

    env = BinanceEnvironment.get_testnet_env()
    client = HttpApiClient(env=env)

    dcm = await DepthCacheManager.create(client, loop, "100K-9BC_BNB", process_depth,␣
↪env=env)

    while True:
        print("doing a sleep")
        await asyncio.sleep(20, loop=loop)


if __name__ == "__main__":
```

(continues on next page)

(continued from previous page)

```
    loop = asyncio.get_event_loop()
    loop.run_until_complete(main())
```

CHAPTER 16

Signing Service

A Service to sign and optionally also broadcast messages for you.

The service holds the private keys of the accounts and supplies a username and password to interact with these accounts.

This client re-uses the binance_chain.messages types. In this case no wallet parameter is required.

This client interacts with the binance-chain-signing-service read the docs there to create our own signing service.

Signing and then broadcasting

```
from binance_chain.messages import NewOrderMsg
from binance_chain.signing.http import HttpApiSigningClient
from binance_chain.constants import TimeInForce, OrderSide, OrderType

signing_client = HttpApiSigningClient('http://localhost:8000', username='sam',␣
↪password='mypass')

# print(client.signing_service_auth())

new_order_msg = NewOrderMsg(
    symbol='ANN-457_BNB',
    order_type=OrderType.LIMIT,
    side=OrderSide.BUY,
    price=0.000396000,
    quantity=10,
    time_in_force=TimeInForce.GOOD_TILL_EXPIRE
)
new_order_hex = signing_client.sign_order(new_order_msg, wallet_name='wallet_1')
```

the *sign_order* method can also take a binance_chain.messages.LimitOrderBuyMsg or binance_chain.messages.LimitOrderSellMsg instance.

This hex can then be broadcast using the normal HTTP Client like so

```
from binance_chain.http import HttpApiClient
from binance_chain.environment import BinanceEnvironment

# initialise with environment that is supported by the signing service wallet
testnet_env = BinanceEnvironment.get_testnet_env()
client = HttpApiClient(env=testnet_env)

res = client.broadcast_hex_msg(new_order_hex['signed_msg'], sync=True)
```

The signing service supports binance_chain.messages types NewOrderMsg, CancelOrderMsg, FreezeMsg, UnFreezeMsg and TransferMsg

Signing and broadcasting in one

To sign and broadcast an order use the *broadcast_order* method. This returns the response from the Binance Chain exchange.

```
from binance_chain.messages import NewOrderMsg
from binance_chain.signing.http import HttpApiSigningClient
from binance_chain.constants import TimeInForce, OrderSide, OrderType

signing_client = HttpApiSigningClient('http://localhost:8000', username='sam',␣
↪password='mypass')

# print(client.signing_service_auth())

new_order_msg = NewOrderMsg(
    symbol='ANN-457_BNB',
    order_type=OrderType.LIMIT,
    side=OrderSide.BUY,
    price=0.000396000,
    quantity=10,
    time_in_force=TimeInForce.GOOD_TILL_EXPIRE
)
res = signing_client.broadcast_order(new_order_msg, wallet_name='wallet_1')
```

CHAPTER 17

Async Signing Service

Like all other libraries there is an async version.

```
from binance_chain.signing.http import AsyncHttpApiSigningClient
from binance_chain.http import AsyncHttpApiClient, PeerType
from binance_chain.environment import BinanceEnvironment
from binance_chain.constants import TimeInForce, OrderSide, OrderType
import asyncio

loop = None

async def main():
    global loop

    # create the client using the classmethod
    signing_client = await AsyncHttpApiSigningClient.create('http://localhost:8000',␣
↪username='sam', password='mypass')

    new_order_msg = NewOrderMsg(
        symbol='ANN-457_BNB',
        order_type=OrderType.LIMIT,
        side=OrderSide.BUY,
        price=0.000396000,
        quantity=10,
        time_in_force=TimeInForce.GOOD_TILL_EXPIRE
    )

    # simply sign the message
    sign_res = await signing_client.sign_order(new_order_msg, wallet_name='wallet_1')

    # or broadcast it as well
    broadcast_res = await signing_client.broadcast_order(new_order_msg, wallet_name=
↪'wallet_1')

    print(json.dumps(await rcp_client.get_abci_info(), indent=2))
```

(continues on next page)

(continued from previous page)

```
    while True:
        print("doing a sleep")
        await asyncio.sleep(20, loop=loop)


if __name__ == "__main__":

    loop = asyncio.get_event_loop()
    loop.run_until_complete(main())
```

CHAPTER 18

Ledger

Sign transactions with your Ledger wallet, supports Ledger Blue, Nano S and Nano X.

Make sure you have registered on Binance Chain with your Ledger address.

Make sure that you have connected your Ledger and are in the Binance Chain app.

Install python-binance-chain with this optional library like so *pip install python-binance-chain[ledger]*

Uses the btchip-python library if having issues installing check their github page

```
from binance_chain.ledger import getDongle, LedgerApp, LedgerWallet
from binance_chain.environment import BinanceEnvironment

dongle = getDongle(debug=True)

testnet_env = BinanceEnvironment.get_testnet_env()
app = LedgerApp(dongle, env=testnet_env)

# get the Ledger Binance app version
print(app.get_version())

# Show your address on the Ledger
print(app.show_address())

# Get your address and public key from the Ledger
print(app.get_address())

# Get your public key from the Ledger
print(app.get_public_key())
```

Create a Wallet to use with the HTTP and Node RPC clients

```
# this will prompt you on your Ledger to confirm the address you want to use
wallet = LedgerWallet(app, env=testnet_env)
```

(continues on next page)

(continued from previous page)

```
# now create messages and sign them with this wallet
from binance_chain.http import HttpApiClient
from binance_chain.messages import NewOrderMsg, OrderType, OrderSide, TimeInForce

client = HttpApiClient(env=testnet_env)
new_order_msg = NewOrderMsg(
    wallet=wallet,
    symbol='ANN-457_BNB',
    order_type=OrderType.LIMIT,
    side=OrderSide.BUY,
    price=0.000396000,
    quantity=10,
    time_in_force=TimeInForce.GOOD_TILL_EXPIRE
)
new_order_res = client.broadcast_msg(new_order_msg, sync=True)

print(new_order_res)
```

CHAPTER 19

Requests and AioHTTP Settings

python-binance-chain uses requests and aiohttp libraries.

You can set custom requests parameters for all API calls when creating any of the http clients.

```
client = HttpApiClient(request_params={"verify": False, "timeout": 20})
```

Check out either the requests documentation or aiohttp documentation for all options.

Proxy Settings

You can use the settings method above

```
proxies = {
    'http': 'http://10.10.1.10:3128',
    'https': 'http://10.10.1.10:1080'
}

# in the Client instantiation
client = HttpApiClient(request_params={'proxies': proxies})
```

Or set an environment variable for your proxy if required to work across all requests.

An example for Linux environments from the requests Proxies documentation is as follows.

```
$ export HTTP_PROXY="http://10.10.1.10:3128"
$ export HTTPS_PROXY="http://10.10.1.10:1080"
```

For Windows environments

```
C:\>set HTTP_PROXY=http://10.10.1.10:3128
C:\>set HTTPS_PROXY=http://10.10.1.10:1080
```

CHAPTER 20

Running Tests

```
git clone https://github.com/sammchardy/python-binance-chain.git
cd python-binance-chain
pip install -r test-requirements.txt

python -m pytest tests/
```

CHAPTER 21

Donate

If this library helped you out feel free to donate.

- ETH: 0xD7a7fDdCfA687073d7cC93E9E51829a727f9fE70
- NEO: AVJB4ZgN7VgSUtArCt94y7ZYT6d5NDfpBo
- LTC: LPC5vw9ajR1YndE1hYVeo3kJ9LdHjcRCUZ
- BTC: 1Dknp6L6oRZrHDECRedihPzx2sSfmvEBys

CHAPTER 22

Thanks

Sipa <*https://github.com/sipa/bech32*> for python reference implementation for Bech32 and segwit addresses

CHAPTER 23

Other Exchanges

If you use Binance check out my python-binance library.

If you use Kucoin check out my python-kucoin library.

If you use IDEX check out my python-idex library.

23.1 Contents

23.1.1 Changelog

v0.1.20 - 2019-06-29

Added

- Multi Transfer Msg option

Fixed

- Response code of 0 for http requests

v0.1.19 - 2019-04-30

Added

- Shuffling of peer nodes in the Pooled HTTP RPC Node client

v0.1.18 - 2019-04-29

Added

- Advanced async Pooled HTTP RPC Node client spreading requests over available peers

Updated

- RPC request id behaviour to work across multiple connections

v0.1.17 - 2019-04-29

Added

- UltraJson ultra fast JSON parser library
- Docs for requests/aiohttp params and proxy support
- more docs and tests

Fixed

- DepthCache close method

v0.1.16 - 2019-04-28

Added

- Vote transaction type
- Simple transaction signing example

Fixed

- Depth Cache doesn't wait for websocket messages before outputting

v0.1.13 - 2019-04-28

Added

- *get_address* function for Ledger hardware wallet
- better error handling and parsing of Ledger hardware wallet responses

v0.1.12 - 2019-04-27

Added

- Ledger Hardware wallet support for signing transactions

v0.1.10 - 2019-04-24

Fixed

- Connecting to secure and insecure websocket connections

v0.1.9 - 2019-04-23

Added

- More test coverage including for python 3.7

Fixed

- Params in async http client for *get_klines*

- coveralls report
- small fixes

v0.1.8 - 2019-04-21

Added

- *get_block_exchange_fee* function to Http API Clients
- memo param to Transfer message
- more tests

Updated

- Remove jsonrpcclient dependency

Fixed

- Use of enums in request params
- Some deprecation warnings

v0.1.7 - 2019-04-18

Updated

- fix package

v0.1.6 - 2019-04-17

Updated

- fix package requirement versions

v0.1.5 - 2019-04-17

Fixed

- signing module imported

v0.1.4 - 2019-04-16

Fixed

- Issue with protobuf file

v0.1.3 - 2019-04-16

Added

- Wallet methods for Binance Signing Service v0.0.2

v0.1.2 - 2019-04-14

Added

- Binance Chain Signing Service Interfaces v0.0.1

Updated

- Cleaned up TransferMsg as from_address is found from wallet instance

v0.1.1 - 2019-04-13

Added

- Broadcast message taking signed hex data

v0.1.0 - 2019-04-11

Added

- Async versions of HTTP Client
- Async version of Node RPC Client
- Node RPC Websocket client
- Async Depth Cache
- Transfer message implementation
- Message broadcast over Node RPC

v0.0.5 - 2019-04-08

Added

- All websocket stream endpoints
- Wallet functions to read account and keep track of transaction sequence
- Support for Testnet and Production environments, along with user defined environment
- Helper classes to create limit buy and sell messages

Updated

- Refactored modules and tidied up message creation and wallets

v0.0.4 - 2019-04-07

Added

- Wallet initialise from private key or mnemonic string
- Create wallet by generating a mnemonic

v0.0.3 - 2019-04-06

Added

- Transaction Broadcasts
- Generated Docs

v0.0.2 - 2019-04-04

Added

- NodeRPC implementation
- Websockets

v0.0.1 - 2019-02-24

- HTTP API Implementation

23.1.2 Binance Chain API

client module

environment module

class binance_chain.environment.**BinanceEnvironment**(*api_url: str = None*, *wss_url: str = None*, *hrp: str = None*)

Bases: object

PROD_ENV = {'api_url': 'https://dex.binance.org', 'hrp': 'bnb', 'wss_url': 'wss://d

TESTNET_ENV = {'api_url': 'https://testnet-dex.binance.org', 'hrp': 'tbnb', 'wss_url

__init__(*api_url: str = None*, *wss_url: str = None*, *hrp: str = None*)

Create custom environment

classmethod get_production_env()

classmethod get_testnet_env()

api_url

wss_url

hrp

hash()

wallet module

messages module

websockets module

node rpc http module

node rpc websockets module

signing service http module

deptch cache module

constants module

class binance_chain.constants.**KlineInterval**
Bases: str, enum.Enum

An enumeration.

ONE_MINUTE = '1m'

THREE_MINUTES = '3m'

FIVE_MINUTES = '5m'

FIFTEEN_MINUTES = '15m'

THIRTY_MINUTES = '30m'

ONE_HOUR = '1h'

TWO_HOURS = '2h'

FOUR_HOURS = '4h'

SIX_HOURS = '6h'

EIGHT_HOURS = '8h'

TWELVE_HOURS = '12h'

ONE_DAY = '1d'

THREE_DAYS = '3d'

ONE_WEEK = '1w'

ONE_MONTH = '1M'

class binance_chain.constants.**OrderStatus**
Bases: str, enum.Enum

An enumeration.

ACK = 'Ack'

PARTIAL_FILL = 'PartialFill'

IOC_NO_FILL = 'IocNoFill'

FULLY_FILL = 'FullyFill'

CANCELED = 'Canceled'

EXPIRED = 'Expired'

FAILED_BLOCKING = 'FailedBlocking'

FAILED_MATCHING = 'FailedMatching'

class binance_chain.constants.**OrderSide**
Bases: int, enum.Enum

An enumeration.

BUY = 1

SELL = 2

class binance_chain.constants.**TimeInForce**
Bases: int, enum.Enum

An enumeration.

GOOD_TILL_EXPIRE = 1

IMMEDIATE_OR_CANCEL = 3

class binance_chain.constants.**TransactionSide**
Bases: str, enum.Enum

An enumeration.

RECEIVE = 'RECEIVE'

SEND = 'SEND'

class binance_chain.constants.**TransactionType**
Bases: str, enum.Enum

An enumeration.

NEW_ORDER = 'NEW_ORDER'

ISSUE_TOKEN = 'ISSUE_TOKEN'

BURN_TOKEN = 'BURN_TOKEN'

LIST_TOKEN = 'LIST_TOKEN'

CANCEL_ORDER = 'CANCEL_ORDER'

FREEZE_TOKEN = 'FREEZE_TOKEN'

UN_FREEZE_TOKEN = 'UN_FREEZE_TOKEN'

TRANSFER = 'TRANSFER'

PROPOSAL = 'PROPOSAL'

VOTE = 'VOTE'

class binance_chain.constants.**OrderType**
Bases: int, enum.Enum

An enumeration.

LIMIT = 2

class binance_chain.constants.**PeerType**
Bases: str, enum.Enum

An enumeration.

NODE = 'node'

WEBSOCKET = 'ws'

class binance_chain.constants.**RpcBroadcastRequestType**
Bases: int, enum.Enum

An enumeration.

SYNC = 1

ASYNC = 2

COMMIT = 3

class binance_chain.constants.**VoteOption**
Bases: int, enum.Enum

An enumeration.

YES = 1

ABSTAIN = 2

NO = 3

NO_WITH_VETO = 4

exceptions module

utils.encode_utils module

binance_chain.utils.encode_utils.**encode_number**(*num: Union[float, decimal.Decimal]*) → int
Encode number multiply by 1e8 (10^8) and round to int

Parameters **num** – number to encode

binance_chain.utils.encode_utils.**varint_encode**(*num*)
Convert number into varint bytes

Parameters **num** – number to encode

23.2 Index

- genindex

Python Module Index

b

Index

Symbols

A

B

C

E

F

G

H

I