raiblocks Documentation

Release 1.0.0

Daniel Dourvaris

Feb 10, 2018

Contents:

This library contains a python wrapper for the RaiBlocks RPC server which tries to make it a little easier to work with by converting RPC responses to native python ones and exposing a pythonic api for making RPC calls.

Also included are utilities such as converting rai/xrb and interesting accounts

CHAPTER 1

RaiBlocks Python Library

This library contains a python wrapper for the RaiBlocks RPC server which tries to make it a little easier to work with by converting RPC responses to native python ones and exposing a pythonic api for making RPC calls.

Also included are utilities such as converting rai/xrb and interesting accounts

1.1 Installation

```
pip install raiblocks
```

1.2 Documentation

https://raiblocks-python.readthedocs.io/

1.3 RPC client

You can browse the available RPC methods list or check the RPC Client API documentation for examples of usage.

```
>>> from raiblocks import RPCClient
>>> rpc = RPCClient('http://localhost:7076')
>>> rpc.version()
{
    'rpc_version': 1,
    'store_version': 10,
    'node_vendor': 'RaiBlocks 9.0'
}
>>> rpc.peers()
{
    '[::ffff:75.171.168.5]:7075': 4,
```

```
    '[::ffff:108.44.38.183]:1032': 4
}
```

1.4 Conversion

```
>>> from raiblocks import convert
>>> convert(12, from_unit='XRB', to_unit='raw')
Decimal('1.2E+31')

>>> convert(0.4, from_unit='krai', to_unit='XRB')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: float values can lead to unexpected
precision loss, please use a Decimal or string
eg. convert('0.4', 'krai', 'XRB')

>>> convert('0.4', from_unit='krai', to_unit='XRB')
Decimal('0.0004')
```

1.5 Known Accounts / Constants

```
>>> from raiblocks import GENESIS_BLOCK_HASH
>>> GENESIS_BLOCK_HASH
'991CF190094C00F0B68E2E5F75F6BEE95A2E0BD93CEAA4A6734DB9F19B728948'
```

```
>>> from raiblocks import KNOWN_ACCOUNT_IDS
>>> KNOWN_ACCOUNT_IDS['xrb_
↪1ipx847tk8o46pwxt5qjdbncjqcbwcc1rrmqnkztrfjy5k7z4imsrata9est']
'Developer Fund'
```

```
>>> from raiblocks import KNOWN_ACCOUNT_NAMES
>>> KNOWN_ACCOUNT_NAMES['Burn']
'xrb_1111111111111111111111111111111111111111111111111111hifc8npp'
```

1.6 Development

1.6.1 Setup

```
virtualenv venv
source venv/bin/activate
pip install -r requirements.pip -r test-requirements.pip
python setup.py develop
```

1.6.2 Running tests

```
# regular
pytest

# coverage
./coverage
```

1.6.3 Building docs

```
cd docs

# generate once
make html

# live building
make live
```

CHAPTER 2

RPC methods

This documents the available methods on the `raiblocks.rpc.RPCClient`

2.1 Account

2.1.1 account_balance

Returns how many RAW is owned and how many have not yet been received by **account** `raiblocks.rpc.RPCClient.account_balance(account)`

2.1.2 account_block_count

Get number of blocks for a specific **account** `raiblocks.rpc.RPCClient.account_block_count(account)`

2.1.3 account_create

Creates a new account, insert next deterministic key in **wallet** `raiblocks.rpc.RPCClient.account_create(wallet, work=True)`

2.1.4 account_get

Get account number for the **public key** `raiblocks.rpc.RPCClient.account_get(key)`

2.1.5 account_history

Reports send/receive information for a **account** `raiblocks.rpc.RPCClient.account_history(account, count)`

2.1.6 account_info

Returns frontier, open block, change representative block, balance, last modified timestamp from local database & block count for **account** `raiblocks.rpc.RPCClient.account_info(account, representative=False, weight=False, pending=False)`

2.1.7 account_key

Get the public key for **account** `raiblocks.rpc.RPCClient.account_key(account)`

2.1.8 account_list

Lists all the accounts inside **wallet** `raiblocks.rpc.RPCClient.account_list(wallet)`

2.1.9 account_move

Moves **accounts** from **source** to **wallet** `raiblocks.rpc.RPCClient.account_move(source, wallet, accounts)`

2.1.10 account_remove

Remove **account** from **wallet** `raiblocks.rpc.RPCClient.account_remove(wallet, account)`

2.1.11 account_representative

Returns the representative for **account** `raiblocks.rpc.RPCClient.account_representative(account)`

2.1.12 account_representative_set

Sets the representative for **account** in **wallet** `raiblocks.rpc.RPCClient.account_representative_set(wallet, account, representative, work=None)`

2.1.13 account_weight

Returns the voting weight for **account** `raiblocks.rpc.RPCClient.account_weight(account)`

2.1.14 accounts_balances

Returns how many RAW is owned and how many have not yet been received by **accounts** list `raiblocks.rpc.RPCClient.accounts_balances(accounts)`

2.1.15 accounts_create

Creates new accounts, insert next deterministic keys in **wallet** up to **count** `raiblocks.rpc.RPCClient.accounts_create(wallet, count, work=True)`

2.1.16 accounts_frontiers

Returns a list of pairs of account and block hash representing the head block for **accounts** list `raiblocks.rpc.RPCClient.accounts_frontiers(accounts)`

2.1.17 accounts_pending

Returns a list of block hashes which have not yet been received by these **accounts** `raiblocks.rpc.RPCClient.accounts_pending(accounts, count=None, threshold=None, source=False)`

2.1.18 block_account

Returns the account containing block `raiblocks.rpc.RPCClient.block_account(hash)`

2.1.19 delegators

Returns a list of pairs of delegator names given **account** a representative and its balance `raiblocks.rpc.RPCClient.delegators(account)`

2.1.20 delegators_count

Get number of delegators for a specific representative **account** `raiblocks.rpc.RPCClient.delegators_count(account)`

2.1.21 frontiers

Returns a list of pairs of account and block hash representing the head block starting at **account** up to **count** `raiblocks.rpc.RPCClient.frontiers(account, count)`

2.1.22 ledger

Returns frontier, open block, change representative block, balance, last modified timestamp from local database & block count starting at **account** up to **count** `raiblocks.rpc.RPCClient.ledger(account, count=None, representative=False, weight=False, pending=False, sorting=False)`

2.1.23 payment_wait

Wait for payment of **amount** to arrive in **account** or until **timeout** milliseconds have elapsed. `raiblocks.rpc.RPCClient.payment_wait(account, amount, timeout)`

2.1.24 pending

Returns a list of pending block hashes with amount more or equal to **threshold** `raiblocks.rpc.RPCClient.pending(account, count=None, threshold=None, source=False)`

2.1.25 receive

Receive pending **block** for **account** in **wallet** `raiblocks.rpc.RPCClient.receive(wallet, account, block, work=None)`

2.1.26 send

Send **amount** from **source** in **wallet** to **destination** `raiblocks.rpc.RPCClient.send(wallet, source, destination, amount, work=None)`

2.1.27 validate_account_number

Check whether **account** is a valid account number `raiblocks.rpc.RPCClient.validate_account_number(account)`

2.2 Block

2.2.1 block

Retrieves a json representation of **block** `raiblocks.rpc.RPCClient.block(hash)`

2.2.2 block_account

Returns the account containing block `raiblocks.rpc.RPCClient.block_account(hash)`

2.2.3 block_count

Reports the number of blocks in the ledger and unchecked synchronizing blocks `raiblocks.rpc.RPCClient.block_count()`

2.2.4 block_count_type

Reports the number of blocks in the ledger by type (send, receive, open, change) `raiblocks.rpc.RPCClient.block_count_type()`

2.2.5 block_create

Creates a json representations of new block based on input data & signed with private key or account in **wallet** for offline signing `raiblocks.rpc.RPCClient.block_create(type, account, wallet=None, representative=None, key=None, destination=None, amount=None, balance=None, previous=None, source=None, work=None)`

2.2.6 blocks

Retrieves a json representations of **blocks** `raiblocks.rpc.RPCClient.blocks(hashes)`

2.2.7 blocks_info

Retrieves a json representations of **blocks** with transaction **amount** & block **account** `raiblocks.rpc.RPCClient.blocks_info(hashes, pending=False, source=False)`

2.2.8 chain

Returns a list of block hashes in the account chain starting at **block** up to **count** `raiblocks.rpc.RPCClient.chain(block, count)`

2.2.9 history

Reports send/receive information for a chain of blocks `raiblocks.rpc.RPCClient.history(hash, count)`

2.2.10 pending_exists

Check whether block is pending by **hash** `raiblocks.rpc.RPCClient.pending_exists(hash)`

2.2.11 process

Publish **block** to the network `raiblocks.rpc.RPCClient.process(block)`

2.2.12 receive

Receive pending **block** for **account** in **wallet** `raiblocks.rpc.RPCClient.receive(wallet, account, block, work=None)`

2.2.13 republish

Rebroadcast blocks starting at **hash** to the network `raiblocks.rpc.RPCClient.republish(hash, count=None, sources=None, destinations=None)`

2.2.14 successors

Returns a list of block hashes in the account chain ending at **block** up to **count** `raiblocks.rpc.RPCClient.successors(block, count)`

2.2.15 unchecked

Returns a list of pairs of unchecked synchronizing block hash and its json representation up to **count** `raiblocks.rpc.RPCClient.unchecked(count=None)`

2.2.16 unchecked_clear

Clear unchecked synchronizing blocks `raiblocks.rpc.RPCClient.unchecked_clear()`

2.2.17 unchecked_get

Retrieves a json representation of unchecked synchronizing block by **hash** `raiblocks.rpc.RPCClient.unchecked_get(hash)`

2.2.18 unchecked_keys

Retrieves unchecked database keys, blocks hashes & a json representations of unchecked pending blocks starting from **key** up to **count** `raiblocks.rpc.RPCClient.unchecked_keys(key=None, count=None)`

2.2.19 work_validate

Check whether **work** is valid for block `raiblocks.rpc.RPCClient.work_validate(work, hash)`

2.3 Global

2.3.1 available_supply

Returns how many rai are in the public supply `raiblocks.rpc.RPCClient.available_supply()`

2.3.2 block_count

Reports the number of blocks in the ledger and unchecked synchronizing blocks `raiblocks.rpc.RPCClient.block_count()`

2.3.3 block_count_type

Reports the number of blocks in the ledger by type (send, receive, open, change) `raiblocks.rpc.RPCClient.block_count_type()`

2.3.4 frontier_count

Reports the number of accounts in the ledger `raiblocks.rpc.RPCClient.frontier_count()`

2.3.5 representatives

Returns a list of pairs of representative and its voting weight `raiblocks.rpc.RPCClient.representatives(count=None, sorting=False)`

2.4 Node

2.4.1 bootstrap

Initialize bootstrap to specific **IP address** and **port** `raiblocks.rpc.RPCClient.bootstrap(address, port)`

2.4.2 bootstrap_any

Initialize multi-connection bootstrap to random peers `raiblocks.rpc.RPCClient.bootstrap_any()`

2.4.3 keepalive

Tells the node to send a keepalive packet to **address**:**port** `raiblocks.rpc.RPCClient.keepalive(address, port)`

2.4.4 peers

Returns a list of pairs of peer IPv6:port and its node network version `raiblocks.rpc.RPCClient.peers()`

2.4.5 receive_minimum

Returns receive minimum for node `raiblocks.rpc.RPCClient.receive_minimum()`

2.4.6 receive_minimum_set

Set **amount** as new receive minimum for node until restart `raiblocks.rpc.RPCClient.receive_minimum_set(amount)`

2.4.7 search_pending_all

Tells the node to look for pending blocks for any account in all available wallets `raiblocks.rpc.RPCClient.search_pending_all()`

2.4.8 stop

Stop the node `raiblocks.rpc.RPCClient.stop()`

2.4.9 unchecked

Returns a list of pairs of unchecked synchronizing block hash and its json representation up to **count** `raiblocks.rpc.RPCClient.unchecked(count=None)`

2.4.10 unchecked_clear

Clear unchecked synchronizing blocks `raiblocks.rpc.RPCClient.unchecked_clear()`

2.4.11 unchecked_get

Retrieves a json representation of unchecked synchronizing block by **hash** `raiblocks.rpc.RPCClient.unchecked_get(hash)`

2.4.12 unchecked_keys

Retrieves unchecked database keys, blocks hashes & a json representations of unchecked pending blocks starting from **key** up to **count** `raiblocks.rpc.RPCClient.unchecked_keys(key=None, count=None)`

2.4.13 version

Returns the node's RPC version `raiblocks.rpc.RPCClient.version()`

2.5 Utility

2.5.1 deterministic_key

Derive deterministic keypair from **seed** based on **index** `raiblocks.rpc.RPCClient.deterministic_key(seed, index)`

2.5.2 key_create

Generates an **adhoc random keypair** `raiblocks.rpc.RPCClient.key_create()`

2.5.3 key_expand

Derive public key and account number from **private key** `raiblocks.rpc.RPCClient.key_expand(key)`

2.5.4 krai_from_raw

Divide a raw amount down by the krai ratio. `raiblocks.rpc.RPCClient.krai_from_raw(amount)`

2.5.5 krai_to_raw

Multiply an krai amount by the krai ratio. `raiblocks.rpc.RPCClient.krai_to_raw(amount)`

2.5.6 mrai_from_raw

Divide a raw amount down by the Mrai ratio. `raiblocks.rpc.RPCClient.mrai_from_raw(amount)`

2.5.7 mrai_to_raw

Multiply an Mrai amount by the Mrai ratio. `raiblocks.rpc.RPCClient.mrai_to_raw(amount)`

2.5.8 rai_from_raw

Divide a raw amount down by the rai ratio. `raiblocks.rpc.RPCClient.rai_from_raw(amount)`

2.5.9 rai_to_raw

Multiply an rai amount by the rai ratio. `raiblocks.rpc.RPCClient.rai_to_raw(amount)`

2.6 Wallet

2.6.1 account_create

Creates a new account, insert next deterministic key in **wallet** `raiblocks.rpc.RPCClient.account_create(wallet, work=True)`

2.6.2 account_list

Lists all the accounts inside **wallet** `raiblocks.rpc.RPCClient.account_list(wallet)`

2.6.3 account_move

Moves **accounts** from **source** to **wallet** `raiblocks.rpc.RPCClient.account_move(source, wallet, accounts)`

2.6.4 account_remove

Remove **account** from **wallet** `raiblocks.rpc.RPCClient.account_remove(wallet, account)`

2.6.5 account_representative_set

Sets the representative for **account** in **wallet** `raiblocks.rpc.RPCClient.account_representative_set(wallet, account, representative, work=None)`

2.6.6 accounts_create

Creates new accounts, insert next deterministic keys in **wallet** up to **count** `raiblocks.rpc.RPCClient.accounts_create(wallet, count, work=True)`

2.6.7 password_change

Changes the password for **wallet** to **password** `raiblocks.rpc.RPCClient.password_change(wallet, password)`

2.6.8 password_enter

Enters the **password** in to **wallet** `raiblocks.rpc.RPCClient.password_enter(wallet, password)`

2.6.9 password_valid

Checks whether the password entered for **wallet** is valid `raiblocks.rpc.RPCClient.password_valid(wallet)`

2.6.10 payment_begin

Begin a new payment session. Searches wallet for an account that's marked as available and has a 0 balance. If one is found, the account number is returned and is marked as unavailable. If no account is found, a new account is created, placed in the wallet, and returned. `raiblocks.rpc.RPCClient.payment_begin(wallet)`

2.6.11 payment_end

End a payment session. Marks the account as available for use in a payment session. `raiblocks.rpc.RPCClient.payment_end(account, wallet)`

2.6.12 payment_init

Marks all accounts in wallet as available for being used as a payment session. `raiblocks.rpc.RPCClient.payment_init(wallet)`

2.6.13 receive

Receive pending **block** for **account** in **wallet** `raiblocks.rpc.RPCClient.receive(wallet, account, block, work=None)`

2.6.14 search_pending

Tells the node to look for pending blocks for any account in **wallet** `raiblocks.rpc.RPCClient.search_pending(wallet)`

2.6.15 send

Send **amount** from **source** in **wallet** to **destination** `raiblocks.rpc.RPCClient.send(wallet, source, destination, amount, work=None)`

2.6.16 wallet_add

Add an adhoc private key **key** to **wallet** `raiblocks.rpc.RPCClient.wallet_add(wallet, key, work=True)`

2.6.17 wallet_balance_total

Returns the sum of all accounts balances in **wallet** `raiblocks.rpc.RPCClient.wallet_balance_total(wallet)`

2.6.18 wallet_balances

Returns how many rai is owned and how many have not yet been received by all accounts in **wallet** `raiblocks.rpc.RPCClient.wallet_balances(wallet)`

2.6.19 wallet_change_seed

Changes seed for **wallet** to **seed** `raiblocks.rpc.RPCClient.wallet_change_seed(wallet, seed)`

2.6.20 wallet_contains

Check whether **wallet** contains **account** `raiblocks.rpc.RPCClient.wallet_contains(wallet, account)`

2.6.21 wallet_create

Creates a new random wallet id `raiblocks.rpc.RPCClient.wallet_create()`

2.6.22 wallet_destroy

Destroys **wallet** and all contained accounts `raiblocks.rpc.RPCClient.wallet_destroy(wallet)`

2.6.23 wallet_export

Return a json representation of **wallet** `raiblocks.rpc.RPCClient.wallet_export(wallet)`

2.6.24 wallet_frontiers

Returns a list of pairs of account and block hash representing the head block starting for accounts from **wallet** `raiblocks.rpc.RPCClient.wallet_frontiers(wallet)`

2.6.25 wallet_key_valid

Returns if a **wallet** key is valid `raiblocks.rpc.RPCClient.wallet_key_valid(wallet)`

2.6.26 wallet_lock

Locks a **wallet** `raiblocks.rpc.RPCClient.wallet_lock(wallet)`

2.6.27 wallet_locked

Checks whether **wallet** is locked `raiblocks.rpc.RPCClient.wallet_locked(wallet)`

2.6.28 wallet_pending

Returns a list of block hashes which have not yet been received by accounts in this **wallet** `raiblocks.rpc.RPCClient.wallet_pending(wallet, count=None, threshold=None, source=False)`

2.6.29 wallet_representative

Returns the default representative for **wallet** `raiblocks.rpc.RPCClient.wallet_representative(wallet)`

2.6.30 wallet_representative_set

Sets the default **representative** for **wallet** `raiblocks.rpc.RPCClient.wallet_representative_set(wallet, representative)`

2.6.31 wallet_republish

Rebroadcast blocks for accounts from **wallet** starting at frontier down to **count** to the network `raiblocks.rpc.RPCClient.wallet_republish(wallet, count)`

2.6.32 wallet_unlock

Unlocks **wallet** using **password** `raiblocks.rpc.RPCClient.wallet_unlock(wallet, password)`

2.7 Work

2.7.1 wallet_work_get

Returns a list of pairs of account and work from **wallet** `raiblocks.rpc.RPCClient.wallet_work_get(wallet)`

2.7.2 work_cancel

Stop generating **work** for block `raiblocks.rpc.RPCClient.work_cancel(hash)`

2.7.3 work_generate

Generates **work** for block `raiblocks.rpc.RPCClient.work_generate(hash)`

2.7.4 work_get

Retrieves work for **account** in **wallet** `raiblocks.rpc.RPCClient.work_get(wallet, account)`

2.7.5 work_peer_add

Add specific **IP address** and **port** as work peer for node until restart `raiblocks.rpc.RPCClient.work_peer_add(address, port)`

2.7.6 work_peers

Retrieve work peers `raiblocks.rpc.RPCClient.work_peers()`

2.7.7 work_peers_clear

Clear work peers node list until restart `raiblocks.rpc.RPCClient.work_peers_clear()`

2.7.8 work_set

Set **work** for **account** in **wallet** `raiblocks.rpc.RPCClient.work_set(wallet, account, work)`

2.7.9 work_validate

Check whether **work** is valid for block `raiblocks.rpc.RPCClient.work_validate(work, hash)`

CHAPTER 3

Utilities

3.1 Conversion tools

For converting between rai/xrb amounts.

The `raiblocks.conversion.convert()` function takes `int`, `Decimal` or `string` arguments (no `float`):

```
>>> from raiblocks import convert
>>> convert(12, from_unit='XRB', to_unit='raw')
Decimal('1.2E+31')

>>> convert(0.4, from_unit='krai', to_unit='XRB')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: float values can lead to unexpected
precision loss, please use a Decimal or string
eg. convert('0.4', 'krai', 'XRB')

>>> convert('0.4', from_unit='krai', to_unit='XRB')
Decimal('0.0004')
```

Warning: Careful not to mix up `'XRB'` and `'xrb'` as they are different units

```
>>> convert(2000000000000000000000000, 'raw', 'XRB')
Decimal('0.000002')
>>> convert(2000000000000000000000000, 'raw', 'xrb')
Decimal('2')
```

For a dict of all available units and their amount in raw:

```
>>> from raiblocks import UNITS_TO_RAW
>>> UNITS_TO_RAW
{'Grai': Decimal('1000000000000000000000000000000000'),
```

```
 'Gxrb': Decimal('1000000000000000000000000000000000'),
 'Mrai': Decimal('1000000000000000000000000000000'),
 'Mxrb': Decimal('1000000000000000000000000000000'),
 'XRB': Decimal('1000000000000000000000000000000'),
 'krai': Decimal('1000000000000000000000000000'),
 'kxrb': Decimal('1000000000000000000000000000'),
 'mrai': Decimal('1000000000000000000000'),
 'mxrb': Decimal('1000000000000000000000'),
 'rai': Decimal('1000000000000000000000'),
 'raw': 1,
 'urai': Decimal('100000000000000000'),
 'uxrb': Decimal('100000000000000000'),
 'xrb': Decimal('1000000000000000000000')}
```

3.2 Known Accounts / Constants

```
>>> from raiblocks import GENESIS_BLOCK_HASH, KNOWN_ACCOUNT_IDS, KNOWN_ACCOUNT_NAMES
>>> KNOWN_ACCOUNT_IDS['xrb_
↪1ipx847tk8o46pwxt5qjdbncjqcbwcc1rrmqnkztrfjy5k7z4imsrata9est']
'Developer Fund'
>>> KNOWN_ACCOUNT_NAMES['Burn']
'xrb_1111111111111111111111111111111111111111111111111hifc8npp'
>>> GENESIS_BLOCK_HASH
'991CF190094C00F0B68E2E5F75F6BEE95A2E0BD93CEAA4A6734DB9F19B728948'
```

CHAPTER 4

raiblocks package

CHAPTER 5

Indices and tables

- genindex
- modindex
- search