raiblocks Documentation

Release 1.0.0

Daniel Dourvaris

Feb 11, 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.

Warning: The RPC client **DOES NOT** handle timeouts or retries automatically since this could lead to unwanted retries of requests causing **double spends**. Keep this in mind when implementing retries.

```
>>> 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 requirements-dev.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
```

1.6.4 Making a release

- Update CHANGELOG.rst
- bumpversion [major|minor|patch]
- python setup.py upload

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

4.1 Submodules

4.2 raiblocks.accounts module

Accounts module

`raiblocks.accounts.KNOWN_ACCOUNT_IDS`: dict of account ids => names eg.

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

`raiblocks.accounts.KNOWN_ACCOUNT_NAMES`: dict of names => account ids

```
>>> KNOWN_ACCOUNT_NAMES['Burn']
'xrb_1111111111111111111111111111111111111111111111111111hifc8npp'
```

`raiblocks.accounts.`**`bytes_to_xrb`**`(`*`value`*`)`

Encodes a hex value to xrb format which uses the base32 algorithm with a custom alphabet: '13456789abcdefghijkmnopqrstuwxyz'

```
>>> xrb_encode(b'deadbeef')
b'ejkp4s54eokpe'
```

`raiblocks.accounts.`**`hex_to_xrb`**`(`*`value`*`)`

Encodes a hex string to xrb format

```
>>> xrb_encode(b'deadbeef')
b'utpuxur'
```

`raiblocks.accounts.`**`xrb_to_bytes`**`(`*`value`*`)`

Encodes an xrb string to bytes

```
>>> xrb_encode(b'ejkp4s54eokpe')
b'deadbeef'
```

raiblocks.accounts.**xrb_to_hex**(*value*)
Encodes an xrb string to hex

```
>>> xrb_encode(b'utpuxur')
b'deadbeef'
```

4.3 raiblocks.blocks module

raiblocks.blocks.**GENESIS_BLOCK_HASH = '991CF190094C00F0B68E2E5F75F6BEE95A2E0BD93CEAA4A6734**
Genesis block hash

4.4 raiblocks.conversion module

Conversion tools for converting xrb

Gxrb = 1000000000000000000000000000000000raw, 10^33

Mxrb = 1000000000000000000000000000000raw, 10^30

kxrb = 1000000000000000000000000000raw, 10^27

xrb = 1000000000000000000000000raw, 10^24

mxrb = 1000000000000000000000raw, 10^21

uxrb = 10000000000000000000raw, 10^18

1 Mxrb used to be also called 1 Mrai 1 xrb is 10^24 raw 1 raw is the smallest possible division

Mrai are XRB 1rai = 1000krai = 1,000,000mrai = 0,000001 XRB

raiblocks.conversion.**convert**(*value*, *from_unit*, *to_unit*)
Converts a value from *from_unit* units to *to_unit* units

Parameters

- **value** (*int or str or decimal.Decimal*) – value to convert
- **from_unit** (*str*) – unit to convert from
- **to_unit** (*str*) – unit to convert to

```
>>> convert(value='1.5', from_unit='xrb', to_unit='krai')
Decimal('0.0015')
```

4.5 raiblocks.rpc module

class raiblocks.rpc.**RPCClient**(*host='http://localhost:7076'*, *session=None*)
Bases: object

RaiBlocks node RPC client

Parameters

- **host** – RPC server host, defaults to *'http://localhost:7076'*
- **session** – optional `requests.Session` session to use for this client

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

account_balance(*account*)

Returns how many RAW is owned and how many have not yet been received by **account**

Parameters **account** (*str*) – Account id to return balance of

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.account_balance(
...     account="xrb_
↪3e3j5tkog48pnny9dmfzj1r16pg8t1e76dz5tmac6iq689wyjfpi00000000"
... )
{
  "balance": 10000,
  "pending": 10000
}
```

account_block_count(*account*)

Get number of blocks for a specific **account**

Parameters **account** (*str*) – Account to get number of blocks for

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.account_block_count(account="xrb_
↪3t6k35gi95xu6tergt6p69ck76ogmitsa8mnijtpxm9fkcm736xtoncuohr3")
19
```

account_create(*wallet*, *work=True*)

Creates a new account, insert next deterministic key in **wallet**

Parameters

- **wallet** (*str*) – Wallet to insert new account into
- **work** (*bool*) – If false, disables work generation after creating account

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.account_create(
...     wallet=
↪"000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F"
... )
"xrb_3e3j5tkog48pnny9dmfzj1r16pg8t1e76dz5tmac6iq689wyjfpi00000000"
```

account_get(*key*)

Get account number for the **public key**

Parameters **key** (*str*) – Public key to get account for

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.account_get(
...     key="3068BB1CA04525BB0E416C485FE6A67FD52540227D267CC8B6E8DA958A7FA039"
... )
"xrb_1e5aqegc1jb7qe964u4adzmcezyo6o146zb8hm6dft8tkp79za3sxwjym5rx"
```

account_history (*account*, *count*)

Reports send/receive information for a **account**

Parameters

- **account** (*str*) – Account to get send/receive information for
- **count** (*int*) – number of blocks to return

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.account_history(
...     account="xrb_
↪3e3j5tkog48pnny9dmfzj1r16pg8t1e76dz5tmac6iq689wyjfpi00000000",
...     count=1
... )
[
    {
      "hash":
↪"000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F",
      "type": "receive",
      "account": "xrb_
↪3e3j5tkog48pnny9dmfzj1r16pg8t1e76dz5tmac6iq689wyjfpi00000000",
      "amount": 10000000000000000000000000000000
    }
]
```

account_info (*account*, *representative=False*, *weight=False*, *pending=False*)

Returns frontier, open block, change representative block, balance, last modified timestamp from local database & block count for **account**

Parameters

- **account** (*str*) – Account to return info for
- **representative** (*bool*) – if True, also returns the representative block
- **weight** (*bool*) – if True, also returns the voting weight
- **pending** (*bool*) – if True, also returns the pending balance

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.account_info(
...     account="xrb_
↪3t6k35gi95xu6tergt6p69ck76ogmitsa8mnijtpxm9fkcm736xtoncuohr3"
... )
{
  "frontier":
↪"FF84533A571D953A596EA401FD41743AC85D04F406E76FDE4408EAED50B473C5",
  "open_block":
↪"991CF190094C00F0B68E2E5F75F6BEE95A2E0BD93CEAA4A6734DB9F19B728948",
  "representative_block":
↪"991CF190094C00F0B68E2E5F75F6BEE95A2E0BD93CEAA4A6734DB9F19B728948",
  "balance": "235580100176034320859259343606608761791",
```

```
  "modified_timestamp": "1501793775",
  "block_count": "33"
}
```

account_key (*account*)
Get the public key for **account**

Parameters **account** (*str*) – Account to get public key for

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.account_key(
...     account="xrb_
↪1e5aqegc1jb7qe964u4adzmcezyo6o146zb8hm6dft8tkp79za3sxwjym5rx"
... )
"3068BB1CA04525BB0E416C485FE6A67FD52540227D267CC8B6E8DA958A7FA039"
```

account_list (*wallet*)
Lists all the accounts inside **wallet**

Parameters **wallet** (*str*) – Wallet to get account list for

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.account_list(
...     wallet=
↪"000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F"
... )
[
    "xrb_3e3j5tkog48pnny9dmfzj1r16pg8t1e76dz5tmac6iq689wyjfpi00000000"
]
```

account_move (*source*, *wallet*, *accounts*)
Moves **accounts** from **source** to **wallet**

Parameters

- **source** (*str*) – wallet to move accounts from
- **wallet** (*str*) – wallet to move accounts to
- **accounts** (*list of str*) – accounts to move

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.account_move(
...     source=
↪"000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F",
...     wallet=
↪"000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F",
...     accounts=[
...         "xrb_3e3j5tkog48pnny9dmfzj1r16pg8t1e76dz5tmac6iq689wyjfpi00000000"
...     ]
... )
True
```

account_remove (*wallet*, *account*)
Remove **account** from **wallet**

Parameters

- **wallet** (*str*) – Wallet to remove account from

- **account** (*str*) – Account to remove

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.account_remove(
...     wallet=
↪"000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F",
...     account="xrb_
↪39a73oy5ungrhxy5z5oao1xso4zo7dmgpjd4u74xcrx3r1w6rtazuouw6qfi"
... )
True
```

account_representative (*account*)

Returns the representative for **account**

Parameters **account** (*str*) – Account to get representative for

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.account_representative(
...     account="xrb_
↪39a73oy5ungrhxy5z5oao1xso4zo7dmgpjd4u74xcrx3r1w6rtazuouw6qfi"
)
"xrb_16u1uufyoig8777y6r8iqjtrw8sg8maqrm36zzcm95jmbd9i9aj5i8abr8u5"
```

account_representative_set (*wallet*, *account*, *representative*, *work=None*)

Sets the representative for **account** in **wallet**

Parameters

- **wallet** (*str*) – Wallet to use for account
- **account** (*str*) – Account to set representative for
- **representative** (*str*) – Representative to set to
- **work** (*str*) – If set, is used as the work for the block

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.account_representative_set(
...     wallet=
↪"000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F",
...     account="xrb_
↪39a73oy5ungrhxy5z5oao1xso4zo7dmgpjd4u74xcrx3r1w6rtazuouw6qfi",
...     representative="xrb_
↪16u1uufyoig8777y6r8iqjtrw8sg8maqrm36zzcm95jmbd9i9aj5i8abr8u5"
... )
"000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F"
```

account_weight (*account*)

Returns the voting weight for **account**

Parameters **account** (*str*) – Account to get voting weight for

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.account_weight(
...     account="xrb_
↪3e3j5tkog48pnny9dmfzj1r16pg8t1e76dz5tmac6iq689wyjfpi00000000"
... )
10000
```

accounts_balances(*accounts*)

Returns how many RAW is owned and how many have not yet been received by **accounts** list

Parameters **accounts** (*list of str*) – list of accounts to return balances for

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.accounts_balances(
...     accounts=[
...         "xrb_3e3j5tkog48pnny9dmfzj1r16pg8t1e76dz5tmac6iq689wyjfpi00000000
↪",
...         "xrb_3i1aq1cchnmbn9x5rsbap8b15akfh7wj7pwskuzi7ahz8oq6cobd99d4r3b7"
...      ]
... )
{
    "xrb_3e3j5tkog48pnny9dmfzj1r16pg8t1e76dz5tmac6iq689wyjfpi00000000": {
        "balance": 10000,
        "pending": 10000
    },
    "xrb_3i1aq1cchnmbn9x5rsbap8b15akfh7wj7pwskuzi7ahz8oq6cobd99d4r3b7": {
        "balance": 10000000,
        "pending": 0
    }
}
```

accounts_create(*wallet*, *count*, *work=True*)

Creates new accounts, insert next deterministic keys in **wallet** up to **count**

Parameters

- **wallet** (*str*) – Wallet to create new accounts in
- **count** (*int*) – Number of accounts to create
- **work** (*bool*) – If false, disables work generation after creating account

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.accounts_create(
...     wallet=
↪"000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F",
...     count=2
... )
[
    "xrb_3e3j5tkog48pnny9dmfzj1r16pg8t1e76dz5tmac6iq689wyjfpi00000000",
    "xrb_1e5aqegc1jb7qe964u4adzmcezyo6o146zb8hm6dft8tkp79za3s00000000"
]
```

accounts_frontiers(*accounts*)

Returns a list of pairs of account and block hash representing the head block for **accounts** list

Parameters **accounts** (*list of str*) – Accounts to return frontier blocks for

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.accounts_frontiers(
...     accounts=[
...         "xrb_3t6k35gi95xu6tergt6p69ck76ogmitsa8mnijtpxm9fkcm736xtoncuohr3
↪",
...         "xrb_3i1aq1cchnmbn9x5rsbap8b15akfh7wj7pwskuzi7ahz8oq6cobd99d4r3b7"
...     ]
... )
```

```
{
    "xrb_3t6k35gi95xu6tergt6p69ck76ogmitsa8mnijtpxm9fkcm736xtoncuohr3":
        "791AF413173EEE674A6FCF633B5DFC0F3C33F397F0DA08E987D9E0741D40D81A",
    "xrb_3i1aq1cchnmbn9x5rsbap8b15akfh7wj7pwskuzi7ahz8oq6cobd99d4r3b7":
        "6A32397F4E95AF025DE29D9BF1ACE864D5404362258E06489FABDBA9DCCC046F"
}
```

accounts_pending(*accounts*, *count=None*, *threshold=None*, *source=False*)

Returns a list of block hashes which have not yet been received by these **accounts**

Parameters

- **accounts** (*list of str*) – Accounts to return list of block hashes for
- **count** (*int*) – Max number of blocks to returns
- **threshold** (*int*) – Minimum amount in raw per block
- **source** (*bool*) – if True returns the source as well

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.accounts_pending(
...     accounts=[
...         "xrb_1111111111111111111111111111111111111111111111111117353trpda
↪",
...         "xrb_3t6k35gi95xu6tergt6p69ck76ogmitsa8mnijtpxm9fkcm736xtoncuohr3"
...     ],
...     count=1
... )
{
    "xrb_1111111111111111111111111111111111111111111111111117353trpda": [
        "142A538F36833D1CC78B94E11C766F75818F8B940771335C6C1B8AB880C5BB1D"
    ],
    "xrb_3t6k35gi95xu6tergt6p69ck76ogmitsa8mnijtpxm9fkcm736xtoncuohr3": [
        "4C1FEEF0BEA7F50BE35489A1233FE002B212DEA554B55B1B470D78BD8F210C74"
    ]
}
```

available_supply()

Returns how many rai are in the public supply

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.available_supply()
10000
```

block(*hash*)

Retrieves a json representation of **block**

Parameters **hash** (*str*) – Hash of block to return representation for

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.block(
...     hash="000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F
↪"
... )
{
    "account": "xrb_
↪3e3j5tkog48pnny9dmfzj1r16pg8t1e76dz5tmac6iq689wyjfpi00000000",
```

```
    "work": "0000000000000000",
    "source":
↪"FA5B51D063BADDF345EFD7EF0D3C5FB115C85B1EF4CDE89D8B7DF3EAF60A04A4",
    "representative": "xrb_
↪3e3j5tkog48pnny9dmfzj1r16pg8t1e76dz5tmac6iq689wyjfpi00000000",
    "signature":
↪"0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪",
    "type": "open"
}
```

block_account(*hash*)

Returns the account containing block

Parameters **hash** (*str*) – Hash of the block to return account for

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.block_account(
...     hash="000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F
↪"
... )
"xrb_3e3j5tkog48pnny9dmfzj1r16pg8t1e76dz5tmac6iq689wyjfpi00000000"
```

block_count()

Reports the number of blocks in the ledger and unchecked synchronizing blocks

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.block_count()
{
  "count": 1000,
  "unchecked": 10
}
```

block_count_type()

Reports the number of blocks in the ledger by type (send, receive, open, change)

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.block_count_type()
{
  "send": 1000,
  "receive": 900,
  "open": 100,
  "change": 50
}
```

block_create(*type*, *account*, *wallet=None*, *representative=None*, *key=None*, *destination=None*, *amount=None*, *balance=None*, *previous=None*, *source=None*, *work=None*)

Creates a json representations of new block based on input data & signed with private key or account in **wallet** for offline signing

Parameters

- **type** (*str*) – Type of block to create one of **open**, **receive**, **change**, **send**
- **account** (*str*) – Account for the signed block
- **wallet** (*str*) – Wallet to use

- **representative** (*str*) – Representative account for **open** and **change** blocks
- **key** (*str*) – Private key to use to open account for **open** blocks
- **destination** (*str*) – Destination account for **send** blocks
- **amount** (*int*) – Amount in raw for **send** blocks
- **balance** (*int*) – Balance in raw of account for **send** blocks
- **previous** (*str*) – Previous block hash for **receive**, **send** and **change** blocks
- **source** (*str*) – Source block for **open** and **receive** blocks
- **work** (*str*) – Work value to use for block from external source

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.block_create(
...     type="open",
...     account="xrb_
↪3kdbxitaj7f6mrir6miiwtw4muhcc58e6tn5st6rfaxsdnb7gr4roudwn951",
...     source=
↪"19D3D919475DEED4696B5D13018151D1AF88B2BD3BCFF048B45031C1F36D1858",
...     representative="xrb_
↪1hza3f7wiiqa7ig3jczyxj5yo86yegcmqk3criaz838j91sxcckpfhbhhra1",
...     key="0000000000000000000000000000000000000000000000000000000000000001"
... )
{
    "block": {
        "account": "xrb_
↪3kdbxitaj7f6mrir6miiwtw4muhcc58e6tn5st6rfaxsdnb7gr4roudwn951",
        "representative": "xrb_
↪1hza3f7wiiqa7ig3jczyxj5yo86yegcmqk3criaz838j91sxcckpfhbhhra1",
        "signature":
↪"5974324F8CC42DA56F62FC212A17886BDCB18DE363D04DA84EEDC99CB4A33919D14A2CF9DE9D534FAA6D0B91D
↪",
        "source":
↪"19D3D919475DEED4696B5D13018151D1AF88B2BD3BCFF048B45031C1F36D1858",
        "type": "open",
        "work": "4ec76c9bda2325ed"
    },
    "hash": "F47B23107E5F34B2CE06F562B5C435DF72A533251CB414C51B2B62A8F63A00E4"
}
```

```
>>> rpc.block_create(
...     type="receive",
...     account="xrb_
↪3kdbxitaj7f6mrir6miiwtw4muhcc58e6tn5st6rfaxsdnb7gr4roudwn951",
...     previous=
↪"F47B23107E5F34B2CE06F562B5C435DF72A533251CB414C51B2B62A8F63A00E4",
...     source=
↪"19D3D919475DEED4696B5D13018151D1AF88B2BD3BCFF048B45031C1F36D1858",
...     wallet=
↪"000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F",
... )
{
    "block": {
        "previous":
↪"F47B23107E5F34B2CE06F562B5C435DF72A533251CB414C51B2B62A8F63A00E4",
        "signature":
↪"A13FD22527771667D5DFF33D69787D734836A3561D8A490C1F4917A05D77EA09860461D5FBFC99246A4EAB562
↪",
```

```
        "source":
↪"19D3D919475DEED4696B5D13018151D1AF88B2BD3BCFF048B45031C1F36D1858",
        "type": "receive",
        "work": "6acb5dd43a38d76a"
    },
    "hash": "314BA8D9057678C1F53371C2DB3026C1FAC01EC8E7802FD9A2E8130FC523429E"
}
```

```
>>> rpc.block_create(
...     type="send",
...     account="xrb_
↪3kdbxitaj7f6mrir6miiwtw4muhcc58e6tn5st6rfaxsdnb7gr4roudwn951",
...     amount=10000000000000000000000000000000,
...     balance=20000000000000000000000000000000,
...     destination="xrb_
↪18gmu6engqhgtjnppqam181o5nfhj4sdtgyhy36dan3jr9spt84rzwmktafc",
...     previous=
↪"314BA8D9057678C1F53371C2DB3026C1FAC01EC8E7802FD9A2E8130FC523429E",
...     wallet=
↪"000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F",
...     work="478563b2d9facfd4",
... )
{
    "block": {
        "balance": "0000007E37BE2022C0914B2680000000",
        "destination": "xrb_
↪18gmu6engqhgtjnppqam181o5nfhj4sdtgyhy36dan3jr9spt84rzwmktafc",
        "previous":
↪"314BA8D9057678C1F53371C2DB3026C1FAC01EC8E7802FD9A2E8130FC523429E",
        "signature":
↪"F19CA177EFA8692C8CBF7478CE3213F56E4A85DF760DA7A9E69141849831F8FD79BA9ED89CEC807B690FB4AA4
↪",
        "type": "send",
        "work": "478563b2d9facfd4"
    },
    "hash": "F958305C0FF0551421D4ABEDCCF302079D020A0A3833E33F185E2B0415D4567A"
}
```

```
>>> rpc.block_create(
...     type="change",
...     account="xrb_
↪3kdbxitaj7f6mrir6miiwtw4muhcc58e6tn5st6rfaxsdnb7gr4roudwn951",
...     representative="xrb_
↪18gmu6engqhgtjnppqam181o5nfhj4sdtgyhy36dan3jr9spt84rzwmktafc",
...     previous=
↪"F958305C0FF0551421D4ABEDCCF302079D020A0A3833E33F185E2B0415D4567A",
...     wallet=
↪"000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F",
... )
{
    "block": {
        "previous":
↪"F958305C0FF0551421D4ABEDCCF302079D020A0A3833E33F185E2B0415D4567A",
        "representative": "xrb_
↪18gmu6engqhgtjnppqam181o5nfhj4sdtgyhy36dan3jr9spt84rzwmktafc",
        "signature":
↪"98B4D56881D9A88B170A6B2976AE21900C26A27F0E2C338D93FDED56183B73D19AA5BEB48E43FCBB8FF8293FD
↪",
```

```
        "type": "change",
        "work": "55e5b7a83edc3f4f"
    },
    "hash": "654FA425CEBFC9E7726089E4EDE7A105462D93DBC915FFB70B50909920A7D286"
}
```

blocks(*hashes*)

Retrieves a json representations of **blocks**

Parameters **hashes** (*list of str*) – List of block hashes to return

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.blocks(
...     hashes=[
↪"000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F"]
... )
{
    "000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F": {
        "account": "xrb_
↪3e3j5tkog48pnny9dmfzj1r16pg8t1e76dz5tmac6iq689wyjfpi00000000",
        "work": "0000000000000000",
        "source":
↪"FA5B51D063BADDF345EFD7EF0D3C5FB115C85B1EF4CDE89D8B7DF3EAF60A04A4",
        "representative": "xrb_
↪3e3j5tkog48pnny9dmfzj1r16pg8t1e76dz5tmac6iq689wyjfpi00000000",
        "signature":
↪"0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪",
        "type": "open"
    }
}
```

blocks_info(*hashes*, *pending=False*, *source=False*)

Retrieves a json representations of **blocks** with transaction **amount** & block **account**

Parameters

- **hashes** (*list of str*) – List of block hashes to return info for
- **pending** (*bool*) – If true, returns pending amount as well
- **source** (*bool*) – If true, returns source account as well

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.blocks_info(hashes=[
↪"000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F"])
{
    "000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F": {
        "block_account": "xrb_
↪3e3j5tkog48pnny9dmfzj1r16pg8t1e76dz5tmac6iq689wyjfpi00000000",
        "amount": "10000000000000000000000000000000",
        "contents": {
            "account": "xrb_
↪3e3j5tkog48pnny9dmfzj1r16pg8t1e76dz5tmac6iq689wyjfpi00000000",
            "work": "0000000000000000",
            "source":
↪"FA5B51D063BADDF345EFD7EF0D3C5FB115C85B1EF4CDE89D8B7DF3EAF60A04A4",
            "representative": "xrb_
↪3e3j5tkog48pnny9dmfzj1r16pg8t1e76dz5tmac6iq689wyjfpi00000000",
```

```
            "signature":
↪"0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪",
            "type": "open"
        }
    }
}
```

bootstrap(*address*, *port*)
Initialize bootstrap to specific **IP address** and **port**

Parameters

- **address** (*str*) – Ip address to bootstrap
- **port** (*int*) – Port to bootstrap

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.bootstrap(address="::ffff:138.201.94.249", port="7075")
True
```

bootstrap_any()
Initialize multi-connection bootstrap to random peers

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.bootstrap_any()
True
```

call(*action*, *params=None*)
Makes an RPC call to the server and returns the json response

Parameters

- **action** (*str*) – RPC method to call
- **params** (*dict*) – Dict of arguments to send with RPC call

Raises *raiblocks.rpc.RPCException*

Raises requests.exceptions.RequestException

```
>>> rpc.call(
...     action='account_balance',
...     params={
...         'account': xrb_
↪3t6k35gi95xu6tergt6p69ck76ogmitsa8mnijtpxm9fkcm736xtoncuohr3'
...     })
{'balance': '325586539664609129644855132177',
 'pending': '230937094000000000000000000000000'}
```

chain(*block*, *count*)
Returns a list of block hashes in the account chain starting at **block** up to **count**

Parameters

- **block** (*str*) – Block hash to start at
- **count** (*int*) – Number of blocks to return up to

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.chain(
...     block=
↪"000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F",
...     count=1
... )
[
    "000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F"
]
```

delegators (*account*)

Returns a list of pairs of delegator names given **account** a representative and its balance

Parameters **account** (*str*) – Account to return delegators for

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.delegators(
...     account="xrb_
↪1111111111111111111111111111111111111111111111117353trpda"
... )
{
    "xrb_13bqhi1cdqq8yb9szneoc38qk899d58i5rcrgdk5mkdm86hekpoez3zxw5sd":
        "500000000000000000000000000000000000",
    "xrb_17k6ug685154an8gri9whhe5kb5z1mf5w6y39gokc1657sh95fegm8ht1zpn":
        "96164797082073000000000000000000000"
}
```

delegators_count (*account*)

Get number of delegators for a specific representative **account**

Parameters **account** (*str*) – Account to get number of delegators for

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.delegators_count(
...     account="xrb_
↪1111111111111111111111111111111111111111111111117353trpda"
... )
2
```

deterministic_key (*seed*, *index*)

Derive deterministic keypair from **seed** based on **index**

Parameters

- **seed** (*str*) – Seed used to get keypair
- **index** (*int*) – Index of the generated keypair

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.deterministic_key(
...     seed="0000000000000000000000000000000000000000000000000000000000000000
↪",
...     index=0
... )
{
  "private": "9F0E444C69F77A49BD0BE89DB92C38FE713E0963165CCA12FAF5712D7657120F
↪",
  "public": "C008B814A7D269A1FA3C6528B19201A24D797912DB9996FF02A1FF356E45552B
↪",
```

```
  "account": "xrb_3i1aq1cchnmbn9x5rsbap8b15akfh7wj7pwskuzi7ahz8oq6cobd99d4r3b7
↪"
}
```

frontier_count()
Reports the number of accounts in the ledger

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.frontier_count()
1000
```

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

Parameters

- **account** (*str*) – Account to get frontier blocks for
- **count** (*int*) – Max amount to return

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.frontiers(
...     account="xrb_
↪1111111111111111111111111111111111111111111111111111hifc8npp",
...     count=1
... )
{
    "xrb_3e3j5tkog48pnny9dmfzj1r16pg8t1e76dz5tmac6iq689wyjfpi00000000":
        "000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F"
}
```

history(*hash*, *count*)
Reports send/receive information for a chain of blocks

Parameters

- **hash** (*str*) – Hash of block to receive history for
- **count** (*int*) – Max number of blocks to return

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.history(
...     hash="000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F
↪",
...     count=1
... )
[
    {
      "hash":
↪"000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F",
      "type": "receive",
      "account": "xrb_
↪3e3j5tkog48pnny9dmfzj1r16pg8t1e76dz5tmac6iq689wyjfpi00000000",
      "amount": "100000000000000000000000000000000"
    }
]
```

keepalive(*address*, *port*)
Tells the node to send a keepalive packet to **address**:**port**

Parameters

- **address** (*str*) – IP address of node to send keepalive packet to
- **port** (*int*) – Port of node to send keepalive packet to

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.keepalive(address="::ffff:192.168.1.1", port=1024)
True
```

key_create()
Generates an **adhoc random keypair**

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.key_create()
{
  "private": "781186FB9EF17DB6E3D1056550D9FAE5D5BBADA6A6BC370E4CBB938B1DC71DA3
↪",
  "public": "3068BB1CA04525BB0E416C485FE6A67FD52540227D267CC8B6E8DA958A7FA039
↪",
  "account": "xrb_1e5aqegc1jb7qe964u4adzmcezyo6o146zb8hm6dft8tkp79za3sxwjym5rx
↪"
}
```

key_expand(*key*)
Derive public key and account number from **private key**

Parameters **key** (*str*) – Private key to generate account and public key of

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.key_expand(
    key="781186FB9EF17DB6E3D1056550D9FAE5D5BBADA6A6BC370E4CBB938B1DC71DA3"
)
{
  "private": "781186FB9EF17DB6E3D1056550D9FAE5D5BBADA6A6BC370E4CBB938B1DC71DA3
↪",
  "public": "3068BB1CA04525BB0E416C485FE6A67FD52540227D267CC8B6E8DA958A7FA039
↪",
  "account": "xrb_1e5aqegc1jb7qe964u4adzmcezyo6o146zb8hm6dft8tkp79za3sxwjym5rx
↪"
}
```

krai_from_raw(*amount*)
Divide a raw amount down by the krai ratio.

Parameters **amount** (*int*) – Amount in raw to convert to krai

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.krai_from_raw(amount=1000000000000000000000000000000)
1
```

krai_to_raw(*amount*)
Multiply an krai amount by the krai ratio.

Parameters **amount** (*int*) – Amount in krai to convert to raw

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.krai_to_raw(amount=1)
1000000000000000000000000000
```

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

Parameters

- **account** (*str*) – Account to return blocks for
- **count** (*int*) – Max number of blocks to return
- **representative** (*bool*) – If true, returns the representative as well
- **weight** (*bool*) – If true, returns the voting weight as well
- **pending** (*bool*) – If true, returns the pending amount as well
- **sorting** (*bool*) – If true, sorts the response by balance

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.ledger(
...     account="xrb_
↪1111111111111111111111111111111111111111111111111111hifc8npp",
...     count=1
... )
{
    "xrb_11119gbh8hb4hj1duf7fdtfyf5s75okzxdgupgpgm1bj78ex3kgy7frt3s9n": {
        "frontier":
↪"E71AF3E9DD86BBD8B4620EFA63E065B34D358CFC091ACB4E103B965F95783321",
        "open_block":
↪"643B77F1ECEFBDBE1CC909872964C1DBBE23A6149BD3CEF2B50B76044659B60F",
        "representative_block":
↪"643B77F1ECEFBDBE1CC909872964C1DBBE23A6149BD3CEF2B50B76044659B60F",
        "balance": 0,
        "modified_timestamp": 1511476234,
        "block_count": 2
    }
}
```

mrai_from_raw(*amount*)
Divide a raw amount down by the Mrai ratio.

Parameters **amount** (*int*) – Amount in raw to convert to Mrai

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.mrai_from_raw(amount=1000000000000000000000000000000)
1
```

mrai_to_raw(*amount*)
Multiply an Mrai amount by the Mrai ratio.

Parameters **amount** (*int*) – Amount in Mrai to convert to raw

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.mrai_to_raw(amount=1)
1000000000000000000000000000000
```

password_change(*wallet*, *password*)
Changes the password for **wallet** to **password**

Parameters

- **wallet** (*str*) – Wallet to change password for
- **password** (*str*) – Password to set

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.password_change(
...     wallet=
↪"000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F",
...     password="test"
... )
True
```

password_enter(*wallet*, *password*)
Enters the **password** in to **wallet**

Parameters

- **wallet** (*str*) – Wallet to enter password for
- **password** (*str*) – Password to enter

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.password_enter(
...     wallet=
↪"000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F",
...     password="test"
... )
True
```

password_valid(*wallet*)
Checks whether the password entered for **wallet** is valid

Parameters **wallet** (*str*) – Wallet to check password for

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.password_valid(
...     wallet=
↪"000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F"
... )
True
```

payment_begin(*wallet*)
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.

Parameters **wallet** (*str*) – Wallet to begin payment in

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.payment_begin(
... wallet="000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F"
```

```
... )
"xrb_3e3j5tkog48pnny9dmfzj1r16pg8t1e76dz5tmac6iq689wyjfpi00000000"
```

payment_end(*account*, *wallet*)
End a payment session. Marks the account as available for use in a payment session.

Parameters

- **account** (*str*) – Account to mark available
- **wallet** (*str*) – Wallet to end payment session for

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.payment_end(
...     account="xrb_
↪3e3j5tkog48pnny9dmfzj1r16pg8t1e76dz5tmac6iq689wyjfpi00000000",
...     wallet=
↪"FFFD1BAEC8EC20814BBB9059B393051AAA8380F9B5A2E6B2489A277D81789EEE"
... )
True
```

payment_init(*wallet*)
Marks all accounts in wallet as available for being used as a payment session.

Parameters **wallet** (*str*) – Wallet to init payment in

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.payment_init(
...     wallet=
↪"000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F"
... )
True
```

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

Parameters

- **account** (*str*) – Account to wait for payment
- **amount** (*int*) – Amount in raw of funds to wait for payment to arrive
- **timeout** (*int*) – Timeout in milliseconds to wait for

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.payment_wait(
...     account="xrb_
↪3e3j5tkog48pnny9dmfzj1r16pg8t1e76dz5tmac6iq689wyjfpi00000000",
...     amount=1,
...     timeout=1000
... )
True
```

peers()
Returns a list of pairs of peer IPv6:port and its node network version

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.peers()
{
    "[::ffff:172.17.0.1]:32841": 3
}
```

pending(*account*, *count=None*, *threshold=None*, *source=False*)

Returns a list of pending block hashes with amount more or equal to **threshold**

Parameters

- **account** (*str*) – Account to get list of pending block hashes for
- **count** (*int*) – Max blocks to return
- **threshold** (*int*) – Minimum amount in raw for blocks
- **source** (*bool*) – If true, returns source address as well

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.pending(
...     account="xrb_
↪1111111111111111111111111111111111111111111111117353trpda"
... )
[
    "000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F"
]
```

```
>>> rpc.pending(
...     account="xrb_
↪1111111111111111111111111111111111111111111111117353trpda",
...     count=1,
...     threshold=1000000000000000000000000
... )
{
    "000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F":
↪"6000000000000000000000000000000"
}
```

pending_exists(*hash*)

Check whether block is pending by **hash**

Parameters **hash** (*str*) – Hash of block to check if pending

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.pending_exists(
    hash="000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F"
)
True
```

process(*block*)

Publish **block** to the network

Parameters **block** (*dict or json*) – Block to publish

Raises *raiblocks.rpc.RPCException*

```
>>> block = {
    "account": "xrb_
↪3e3j5tkog48pnny9dmfzj1r16pg8t1e76dz5tmac6iq689wyjfpi00000000",
```

```
    "work": "0000000000000000",
    "source":
↪"FA5B51D063BADDF345EFD7EF0D3C5FB115C85B1EF4CDE89D8B7DF3EAF60A04A4",
    "representative": "xrb_
↪3e3j5tkog48pnny9dmfzj1r16pg8t1e76dz5tmac6iq689wyjfpi00000000",
    "signature":
↪"0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪",
    "type": "open"
}
```

```
>>> rpc.process(block=block)
"42A723D2B60462BF7C9A003FE9A70057D3A6355CA5F1D0A57581000000000000"
```

```
>>> rpc.process(json.dumps(block))
"42A723D2B60462BF7C9A003FE9A70057D3A6355CA5F1D0A57581000000000000"
```

rai_from_raw(*amount*)

Divide a raw amount down by the rai ratio.

Parameters **amount** (*int*) – Amount in raw to convert to rai

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.rai_from_raw(amount=1000000000000000000000000)
1
```

rai_to_raw(*amount*)

Multiply an rai amount by the rai ratio.

Parameters **amount** (*int*) – Amount in rai to convert to raw

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.rai_to_raw(amount=1)
1000000000000000000000000
```

receive(*wallet*, *account*, *block*, *work=None*)

Receive pending **block** for **account** in **wallet**

Parameters

- **wallet** (*str*) – Wallet of account to receive block for
- **account** (*str*) – Account to receive block for
- **block** (*str*) – Block hash to receive
- **work** (*str*) – If set, uses this work for the receive block

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.receive(
...     wallet=
↪"000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F",
...     account="xrb_
↪3e3j5tkog48pnny9dmfzj1r16pg8t1e76dz5tmac6iq689wyjfpi00000000",
...     block=
↪"53EAA25CE28FA0E6D55EA9704B32604A736966255948594D55CBB05267CECD48",
...     work="12041e830ad10de1"
```

```
... )
"EE5286AB32F580AB65FD84A69E107C69FBEB571DEC4D99297E19E3FA5529547B"
```

receive_minimum()
Returns receive minimum for node

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.receive_minimum()
1000000000000000000000000
```

receive_minimum_set(*amount*)
Set **amount** as new receive minimum for node until restart

Parameters **amount** (*int*) – Amount in raw to set as minimum to receive

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.receive_minimum_set(amount=1000000000000000000000000000000)
True
```

representatives(*count=None*, *sorting=False*)
Returns a list of pairs of representative and its voting weight

Parameters

- **count** (*int*) – Max amount of representatives to return
- **sorting** (*bool*) – If true, sorts by weight

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.representatives()
{
    "xrb_1111111111111111111111111111111111111111111111117353trpda":
        382237232706017000000000000000000000,
    "xrb_1111111111111111111111111111111111111111111111111awsq94gtecn":
        309999999999999999999999000000,
    "xrb_114nk4rwjctu6n6tr6g6ps61g1w3hdpjxfas4xj1tq6i8jyomc5d858xr1xi":
        0
}
```

republish(*hash*, *count=None*, *sources=None*, *destinations=None*)
Rebroadcast blocks starting at **hash** to the network

Parameters

- **hash** (*str*) – Hash of block to start rebroadcasting from
- **count** (*int*) – Max number of blocks to rebroadcast
- **sources** (*int*) – If set, additionally rebroadcasts source chain blocks for receive/open up to **sources** depth
- **destinations** (*int*) – If set, additionally rebroadcasts destination chain blocks for receive/open up to **destinations** depth

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.republish(
...     hash="991CF190094C00F0B68E2E5F75F6BEE95A2E0BD93CEAA4A6734DB9F19B728948
↪"
```

```
... )
[
    "991CF190094C00F0B68E2E5F75F6BEE95A2E0BD93CEAA4A6734DB9F19B728948",
    "A170D51B94E00371ACE76E35AC81DC9405D5D04D4CEBC399AEACE07AE05DD293"
]
```

search_pending(*wallet*)
Tells the node to look for pending blocks for any account in **wallet**

Parameters **wallet** (*str*) – Wallet to search for pending blocks

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.search_pending(
...     wallet=
↪"000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F"
... )
True
```

search_pending_all()
Tells the node to look for pending blocks for any account in all available wallets

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.search_pending_all()
True
```

send(*wallet*, *source*, *destination*, *amount*, *work=None*)
Send **amount** from **source** in **wallet** to **destination**

Parameters

- **wallet** (*str*) – Wallet of account used to send funds
- **source** (*str*) – Account to send funds from
- **destination** (*str*) – Account to send funds to
- **amount** (*int*) – Amount in raw to send
- **work** (*str*) – If set, uses this work for the block

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.send(
...     wallet=
↪"000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F",
...     source="xrb_
↪3e3j5tkog48pnny9dmfzj1r16pg8t1e76dz5tmac6iq689wyjfpi00000000",
...     destination="xrb_
↪3e3j5tkog48pnny9dmfzj1r16pg8t1e76dz5tmac6iq689wyjfpi00000000",
...     amount=1000000,
...     work="2bf29ef00786a6bc"
... )
"000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F"
```

stop()
Stop the node

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.stop()
True
```

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

Parameters

- **block** (*str*) – Hash of block to start returning successors for
- **count** (*int*) – Max number of successor blocks to return

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.successors(
...     block=
↪"991CF190094C00F0B68E2E5F75F6BEE95A2E0BD93CEAA4A6734DB9F19B728948",
...     count=1
... )
[
    "A170D51B94E00371ACE76E35AC81DC9405D5D04D4CEBC399AEACE07AE05DD293"
]
```

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

Parameters **count** (*int*) – Max amount of unchecked blocks to return

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.unchecked(count=1)
{
    "000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F": {
        "account": "xrb_
↪3e3j5tkog48pnny9dmfzj1r16pg8t1e76dz5tmac6iq689wyjfpi00000000",
        "work": "0000000000000000",
        "source":
↪"FA5B51D063BADDF345EFD7EF0D3C5FB115C85B1EF4CDE89D8B7DF3EAF60A04A4",
        "representative": "xrb_
↪3e3j5tkog48pnny9dmfzj1r16pg8t1e76dz5tmac6iq689wyjfpi00000000",
        "signature":
↪"0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪",
        "type": "open"
    }
}
```

unchecked_clear()
Clear unchecked synchronizing blocks

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.unchecked_clear()
True
```

unchecked_get(*hash*)
Retrieves a json representation of unchecked synchronizing block by **hash**

Parameters **hash** (*str*) – Hash of unchecked block to get

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.unchecked_get(
...     hash="000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F
↪"
... )
{
    "account": "xrb_
↪3e3j5tkog48pnny9dmfzj1r16pg8t1e76dz5tmac6iq689wyjfpi00000000",
    "work": "0000000000000000",
    "source":
↪"FA5B51D063BADDF345EFD7EF0D3C5FB115C85B1EF4CDE89D8B7DF3EAF60A04A4",
    "representative": "xrb_
↪3e3j5tkog48pnny9dmfzj1r16pg8t1e76dz5tmac6iq689wyjfpi00000000",
    "signature":
↪"0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪",
    "type": "open"
}
```

unchecked_keys (*key=None*, *count=None*)

Retrieves unchecked database keys, blocks hashes & a json representations of unchecked pending blocks starting from **key** up to **count**

Parameters

- **key** (*str*) – Starting key to return unchecked keys for
- **count** (*int*) – Max number of keys/blocks to return

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.unchecked_keys(
...     key="FA5B51D063BADDF345EFD7EF0D3C5FB115C85B1EF4CDE89D8B7DF3EAF60A04A4
↪",
...     count=1
... )
[
    {
        "key":
↪"FA5B51D063BADDF345EFD7EF0D3C5FB115C85B1EF4CDE89D8B7DF3EAF60A04A4",
        "hash":
↪"000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F",
        "contents": {
            "account": "xrb_
↪3e3j5tkog48pnny9dmfzj1r16pg8t1e76dz5tmac6iq689wyjfpi00000000",
            "work": "0000000000000000",
            "source":
↪"FA5B51D063BADDF345EFD7EF0D3C5FB115C85B1EF4CDE89D8B7DF3EAF60A04A4",
            "representative": "xrb_
↪3e3j5tkog48pnny9dmfzj1r16pg8t1e76dz5tmac6iq689wyjfpi00000000",
            "signature":
↪"0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪",
            "type": "open"
        }
    }
]
```

validate_account_number (*account*)

Check whether **account** is a valid account number

Parameters **account** (*str*) – Account number to check

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.validate_account_number(
...     account="xrb_
↪3e3j5tkog48pnny9dmfzj1r16pg8t1e76dz5tmac6iq689wyjfpi00000000"
... )
True
```

version()

Returns the node's RPC version

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.version()
{
    "rpc_version": 1,
    "store_version": 10,
    "node_vendor": "RaiBlocks 9.0"
}
```

wallet_add(*wallet*, *key*, *work=True*)

Add an adhoc private key **key** to **wallet**

Parameters

- **wallet** (*str*) – Wallet to add private key to
- **key** (*str*) – Private key to add
- **work** (*bool*) – If false, disables work generation

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.wallet_add(
...     wallet=
↪"000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F",
...     key="34F0A37AAD20F4A260F0A5B3CB3D7FB50673212263E58A380BC10474BB039CE4"
... )
"xrb_3e3j5tkog48pnny9dmfzj1r16pg8t1e76dz5tmac6iq689wyjfpi00000000"
```

wallet_balance_total(*wallet*)

Returns the sum of all accounts balances in **wallet**

Parameters **wallet** (*str*) – Wallet to return sum of balances for

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.wallet_balance_total(
...     wallet=
↪"000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F"
... )
{
  "balance": 10000,
  "pending": 10000
}
```

wallet_balances(*wallet*)

Returns how many rai is owned and how many have not yet been received by all accounts in **wallet**

Parameters **wallet** (*str*) – Wallet to return balances for

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.wallet_balances(
...     wallet=
↪"000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F"
... )
{
    "xrb_3e3j5tkog48pnny9dmfzj1r16pg8t1e76dz5tmac6iq689wyjfpi00000000": {
        "balance": 10000,
        "pending": 10000
    }
}
```

wallet_change_seed(*wallet*, *seed*)

Changes seed for **wallet** to **seed**

Parameters

- **wallet** (*str*) – Wallet to change seed for
- **seed** (*str*) – Seed to change wallet to

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.wallet_change_seed(
...     wallet=
↪"000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F",
...     seed="74F2B37AAD20F4A260F0A5B3CB3D7FB51673212263E58A380BC10474BB039CEE
↪"
... )
True
```

wallet_contains(*wallet*, *account*)

Check whether **wallet** contains **account**

Parameters

- **wallet** (*str*) – Wallet to check contains **account**
- **account** (*str*) – Account to check exists in **wallet**

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.wallet_contains(
...     wallet=
↪"000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F",
...     account="xrb_
↪3e3j5tkog48pnny9dmfzj1r16pg8t1e76dz5tmac6iq689wyjfpi00000000"
... )
True
```

wallet_create()

Creates a new random wallet id

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.wallet_create()
"000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F"
```

wallet_destroy(*wallet*)

Destroys **wallet** and all contained accounts

Parameters **wallet** (*str*) – Wallet to destroy

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.wallet_destroy(
...     wallet=
↪"000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F"
... )
True
```

wallet_export(*wallet*)

Return a json representation of **wallet**

Parameters **wallet** (*str*) – Wallet to export

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.wallet_export(wallet=
↪"000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F")
{
    "0000000000000000000000000000000000000000000000000000000000000000":
↪"0000000000000000000000000000000000000000000000000000000000000001"
}
```

wallet_frontiers(*wallet*)

Returns a list of pairs of account and block hash representing the head block starting for accounts from **wallet**

Parameters **wallet** (*str*) – Wallet to return frontiers for

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.wallet_frontiers(
...     wallet=
↪"000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F"
... )
{
    "xrb_3e3j5tkog48pnny9dmfzj1r16pg8t1e76dz5tmac6iq689wyjfpi00000000":
↪"000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F"
}
```

wallet_key_valid(*wallet*)

Returns if a **wallet** key is valid

Parameters **wallet** (*str*) – Wallet to check key is valid

```
>>> rpc.wallet_key_valid(
...     wallet=
↪"000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F"
... )
True
```

wallet_lock(*wallet*)

Locks a **wallet**

Parameters **wallet** (*str*) – Wallet to lock

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.wallet_lock(
...     wallet=
↪"000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F"
```

```
... )
True
```

wallet_locked(*wallet*)
Checks whether **wallet** is locked

Parameters **wallet** (*str*) – Wallet to check if locked

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.wallet_locked(
...     wallet=
↪"000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F"
... )
False
```

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

Parameters

- **wallet** (*str*) – Wallet to get list of pending block hashes for
- **count** (*int*) – Max amount of blocks to return
- **threshold** (*int*) – Minimum amount in raw per block
- **source** (*bool*) – If true, returns source account as well

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.wallet_pending(
...     wallet=
↪"000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F",
...     count=1
... )
{
    "xrb_1111111111111111111111111111111111111111111111111117353trpda": [
        "142A538F36833D1CC78B94E11C766F75818F8B940771335C6C1B8AB880C5BB1D"
    ],
    "xrb_3t6k35gi95xu6tergt6p69ck76ogmitsa8mnijtpxm9fkcm736xtoncuohr3": [
        "4C1FEEF0BEA7F50BE35489A1233FE002B212DEA554B55B1B470D78BD8F210C74"
    ]
}
```

wallet_representative(*wallet*)
Returns the default representative for **wallet**

Parameters **wallet** (*str*) – Wallet to get default representative account for

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.wallet_representative(
...     wallet=
↪"000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F"
... )
"xrb_3e3j5tkog48pnny9dmfzj1r16pg8t1e76dz5tmac6iq689wyjfpi00000000"
```

wallet_representative_set(*wallet*, *representative*)
Sets the default **representative** for **wallet**

Parameters

- **wallet** (*str*) – Wallet to set default representative account for
- **representative** (*str*) – Representative account to set for **wallet**

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.wallet_representative_set(
...     wallet=
↪"000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F",
...     representative="xrb_
↪3e3j5tkog48pnny9dmfzj1r16pg8t1e76dz5tmac6iq689wyjfpi00000000"
... )
True
```

wallet_republish (*wallet*, *count*)

Rebroadcast blocks for accounts from **wallet** starting at frontier down to **count** to the network

Parameters

- **wallet** (*str*) – Wallet to rebroadcast blocks for
- **count** (*int*) – Max amount of blocks to rebroadcast since frontier block

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.wallet_republish(
...     wallet=
↪"000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F",
...     count=2
... )
[
    "991CF190094C00F0B68E2E5F75F6BEE95A2E0BD93CEAA4A6734DB9F19B728948",
    "A170D51B94E00371ACE76E35AC81DC9405D5D04D4CEBC399AEACE07AE05DD293",
    "90D0C16AC92DD35814E84BFBCC739A039615D0A42A76EF44ADAEF1D99E9F8A35"
]
```

wallet_unlock (*wallet*, *password*)

Unlocks **wallet** using **password**

Parameters

- **wallet** (*str*) – Wallet to unlock
- **password** (*str*) – Password to enter

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.wallet_unlock(
...     wallet=
↪"000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F",
...     password="test"
... )
True
```

wallet_work_get (*wallet*)

Returns a list of pairs of account and work from **wallet**

Parameters **wallet** (*str*) – Wallet to return work for

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.wallet_work_get(
...     wallet=
↪"000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F"
... )
{
    "xrb_1111111111111111111111111111111111111111111111111111hifc8npp":
        "432e5cf728c90f4f"
}
```

work_cancel(*hash*)
Stop generating **work** for block

Parameters **hash** (*str*) – Hash to stop generating work for

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.work_cancel(
...     hash="718CC2121C3E641059BC1C2CFC45666C99E8AE922F7A807B7D07B62C995D79E2
↪"
... )
True
```

work_generate(*hash*)
Generates **work** for block

Parameters **hash** (*str*) – Hash to start generating **work** for

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.work_generate(
...     hash="718CC2121C3E641059BC1C2CFC45666C99E8AE922F7A807B7D07B62C995D79E2
↪"
... )
"2bf29ef00786a6bc"
```

work_get(*wallet*, *account*)
Retrieves work for **account** in **wallet**

Parameters

- **wallet** (*str*) – Wallet to get account work for
- **account** (*str*) – Account to get work for

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.work_get(
...     wallet=
↪"000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F",
...     account="xrb_
↪1111111111111111111111111111111111111111111111111111hifc8npp"
... )
"432e5cf728c90f4f"
```

work_peer_add(*address*, *port*)
Add specific **IP address** and **port** as work peer for node until restart

Parameters

- **address** (*str*) – IP address of work peer to add
- **port** (*int*) – Port work peer to add

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.work_peer_add(address="::ffff:172.17.0.1", port="7076")
True
```

work_peers()
Retrieve work peers

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.work_peers()
[
    "::ffff:172.17.0.1:7076"
]
```

work_peers_clear()
Clear work peers node list until restart

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.work_peers_clear()
True
```

work_set(*wallet*, *account*, *work*)
Set **work** for **account** in **wallet**

Parameters

- **wallet** (*str*) – Wallet to set work for account for
- **account** (*str*) – Account to set work for
- **work** (*str*) – Work to set for account in wallet

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.work_set(
...     wallet=
↪"000D1BAEC8EC208142C99059B393051BAC8380F9B5A2E6B2489A277D81789F3F",
...     account="xrb_
↪1111111111111111111111111111111111111111111111111hifc8npp",
...     work="0000000000000000"
... )
True
```

work_validate(*work*, *hash*)
Check whether **work** is valid for block

Parameters

- **work** (*str*) – Work to validate
- **hash** (*str*) – Hash of block to validate work for

Raises *raiblocks.rpc.RPCException*

```
>>> rpc.work_validate(
...     work="2bf29ef00786a6bc",
...     hash="718CC2121C3E641059BC1C2CFC45666C99E8AE922F7A807B7D07B62C995D79E2
↪"
... )
True
```

exception raiblocks.rpc.**RPCException**
Bases: exceptions.Exception

Base class for RPC errors

raiblocks.rpc.**doc_metadata**(*categories*)
Decorator to add doc metadata for docs generation

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

r

Index

P

R

S

U

V

W

X