Cardano Python Module

Release 0.8.2

Michał Sałaban

Jan 27, 2022

CONTENTS:

Welcome to the documentation for the `cardano` Python module.

The aim of this project is to offer a set of tools for interacting with Cardano blockchain platform in Python. It provides higher level classes representing objects from the Cardano environment, like wallets, addresses, transactions.

Currently it operates over REST protocol offered by the `cardano-wallet` binary, however, a forward compatibility for implementing other backends is one of the key features.

Project homepage: https://github.com/emesik/cardano-python

CHAPTER

ONE

QUICK START

This quick start tutorial will guide you through the first steps of connecting to the Cardano wallet software. We assume:

1. You have installed and started cardano-node. (If there's no package for your OS, you may consider installing a Docker image, however many of the tutorials out there are out of date.)
2. The node has synchronized the blockchain.
3. You have installed and started cardano-wallet.
4. The wallet software is connected to the node.
5. You know how to use CLI (*command line interface*).
6. You have some experience with Python.

1.1 Use testnet for your own safety

The testnet is another Cardano network where worthless coins circulate and where, as the name suggests, all tests are supposed to be run. It's also a place for early deployment of future features of the platform itself.

> **Warning:** **Please run all tests on testnet.** The code presented in these docs will perform the requested operations right away, without asking for confirmation. This is live code, not a wallet application that makes sure the user has not made a mistake. **Running on the mainnet, if you make a mistake, you may lose money.**

1.2 Connect to the wallet

For brevity, the following example assumes thet you have an existing wallet of id `eff9cc89621111677a501493ace8c3f05608c0ce` and the `cardano-wallet` is listening locally on port 8090. In the following chapter you'll also learn how to create a new wallet from seed.

```
In [1]: from cardano.wallet import Wallet

In [2]: from cardano.backends.walletrest import WalletREST

In [3]: wal = Wallet("eff9cc89621111677a501493ace8c3f05608c0ce",␣
↪backend=WalletREST(port=8090))

In [4]: wal.sync_progress()
Out[4]: 1.0
```

(continues on next page)

(continued from previous page)

```
In [5]: wal.balance()
Out[5]: Balance(total=Decimal('998.831199'), available=Decimal('998.831199'),␣
↪reward=Decimal('0.000000'))
```

Congratulations! You have connected to the wallet. You may now proceed to the next section, which will tell you about *interaction with wallet*.

CHAPTER

TWO

WORKING WITH WALLETS

The example presented in the *quickstart section* will certainly fail in your environment, as you don't have the same wallet created yet. In order to set up a new wallet, you need to learn first about two kinds of objects:

1. `Wallet` which represents a single Cardano wallet and allows for operations like balance retrieval, searching through the transaction history and spending funds. Wallets are identified by unique ID which is deterministically derived from the seed (mnemonic phrase), so each time you delete and create a wallet again, it receives the same ID.
2. `WalletService` which is responsible for creating new and listing or retrieving existing wallets.
3. Backend, which represents the underlying service layer. At the moment the only backend available is the REST API provided by `cardano-wallet`, represented by the `WalletREST` objects.

Note: Remember that creating or deleting a wallet will not record any information on the blockchain. If the wallet you're creating existed before, you'll be presented its' entire history. Likweise, if you delete a wallet, you'll be able to create it again in this or any other software and claim the funds or see historical transactions.

2.1 Creating wallets

Let's assume your backend doesn't know anything about the wallet `eff9cc89621111677a501493ace8c3f05608c0ce`, which is exactly your starting scenario. In order to obtain that wallet object, you'd have first to create it:

```
In [1]: from cardano.wallet import WalletService

In [2]: from cardano.backends.walletrest import WalletREST

In [3]: ws = WalletService(WalletREST(port=8090))

In [4]: wal = ws.create_wallet(
        name="test wallet",
        mnemonic="resist render west spin antique wild gossip thing syrup network risk␣
↪gospel seek drop receive",
        passphrase="xxx",
        )

In [4]: wal.sync_progress()
Out[4]: 0.05
```

(continues on next page)

(continued from previous page)

```
In [5]: wal.balance()
Out[5]: Balance(total=Decimal('0.000000'), available=Decimal('0.000000'), reward=Decimal(
↪'0.000000'))
```

Even though this wallet may contain some funds (on *testnet*), right after creation the balance will be null and the transaction history empty. This is because of ongoing sync process which scans the entire blockchain for transaction history.

2.1.1 Balance tuple

The balance returned by `wal.balance()` (as well as balances of native assets) is a subclass of `collections.namedtuple`. It consists of three elements:

0. **`total` — indicating the total amount of funds in the wallet, without going into too much** details.
1. **`available` — the amount of funds without staking rewards, might be also considered as the** principal paid to the wallet and used for staking.
2. `reward` — the amount received as staking interest.

Hence, to just get the full balance, you may use `wal.balance().total`.

2.1.2 Sync progress

The value returned by `.sync_progress()` is a float number that represents how advanced the synchronization process is. It starts from `0.0` and goes up to `1.0` which says the wallet is up to date with the blockchain. A simple synchronization wait loop might look like the following:

```
In [6]: import time

In [7]: while wal.sync_progress() < 1.0:
            time.sleep(1)
```

Depending on your conditions, you may use other sleep period value. Just remember to call it within the loop, as it releases CPU time to other processes instead of constantly bombarding your REST API with requests.

2.2 Retrieving existing wallets

In case your backend already knows about the wallet, you may use much simpler approach:

```
In [1]: from cardano.wallet import Wallet

In [2]: from cardano.backends.walletrest import WalletREST

In [3]: wal = Wallet("eff9cc89621111677a501493ace8c3f05608c0ce",␣
↪backend=WalletREST(port=8090))

In [4]: wal.sync_progress()
Out[4]: 1.0
```

(continues on next page)

(continued from previous page)

```
In [5]: wal.balance()
Out[5]: Balance(total=Decimal('998.831199'), available=Decimal('998.831199'),␣
↪reward=Decimal('0.000000'))
```

Note: Although the backend is a required argument right now, the example passes it to the contstructor like it was optional. This is because in the near future some offline functionality will be added to the `Wallet` class and initialization without backend will be available.

2.3 Removing wallets

This is a trivial operation:

```
In [6]: wal.delete()
```

After that, if you try to use the wallet object again, the *cardano.backends.walletrest.exceptions.NotFound* exception will be raised.

2.4 API reference

class cardano.wallet.**Wallet**(*wid*, *backend*, *passphrase=None*)

Represents a single wallet. Allows for browsing the history, checking balance and spending funds.

Parameters

- **wid** – the wallet ID
- **backend** – the backend used to handle the underlying service layer
- **passphrase** – the passphrase protecting the wallet's spending functionality, not required for read-only operations. It will be stored for the entire lifetime of the object in `.passphrase` field. It might be also provided for each individual spend operation, then it will be discarded after use.

addresses(*with_usage=False*)

Returns full list of already generated addresses.

Parameters **with_usage** – A `bool` indicating whether to retrieve used/unused address status too.

Return type `list` of *Address* objects when `with_usage == False` and of (`Address`, `bool`) tuples otherwise.

assets()

Returns the balance of native assets.

Return type `dict` of `AssetID`: *Balance* pairs

balance()

Returns the *Balance* of the wallet.

delete()

Deletes the wallet from the backend. It doesn't wipe the funds; the wallet may be restored later on, using the mnemonic phrase.

estimate_fee(*destinations*, *metadata=None*)
Estimates the fee for a potential transaction to specified destinations and carrying optional metadata. Returns a tuple of estimated minimum and maximum fee, in ADA.

Parameters

- **destinations** – a list of *Address* and amount pairs `[(address, amount), ...]`
- **metadata** – metadata to be sent, as *Metadata* instance od `dict` mapping `int` keys to values of acceptable types

Return type (`Decimal`, `Decimal`)

first_unused_address()
Returns the first unused address. **There is no internal pointer and the result is based on blockchain and mempool state only, and their interpretation by the backend**, so multiple subsequent calls will return the same address if no transfer is received between them.

stake(*pool*, *passphrase=None*)
Stakes all wallet balance at the given pool.

Parameters

- **pool** – The pool to stake ADA at
- **passphrase** – the passphrase to the wallet. It takes precedence over *self.passphrase* and is discarded after use. If neither *self.passphrase* nor *passphrase* is set, a *MissingPassphrase* exception will be raised.

Return type *Transaction*

stake_pools(*stake=None*)
Returns a list of known stake pools ordered by descending rewards.

Parameters **stake** (`Decimal`) – The amount of ADA to be staked. Optional. If omitted, the wallet's total balance will be used instead.

Return type `list`

staking_status()
Returns information about staking status.

Return type *StakingStatus*

sync_progress()
Returns the progress of synchronization with the blockchain. The value is `float` ranging from `0.0` to `1.0`.

transfer(*address*, *amount*, *assets=None*, *metadata=None*, *allow_withdrawal=False*, *ttl=None*, *passphrase=None*)
Sends a transfer from the wallet. Returns the resulting transaction.

Parameters

- **address** – destination *Address* or subtype
- **amount** – amount to send
- **assets** – a sequence of *AssetID* and quantity pairs
- **metadata** – metadata to be sent, as *Metadata* instance od `dict` mapping `int` keys to values of acceptable types
- **allow_withdrawal** – Allow withdrawing staking rewards to cover the transaction amount or fee.

- **ttl** – Time To Live in seconds. After TTL has lapsed the nodes give up on broadcasting the transaction. Leave *None* to use the default value.
- **passphrase** – the passphrase to the wallet. It takes precedence over *self.passphrase* and is discarded after use. If neither *self.passphrase* nor *passphrase* is set, a *MissingPassphrase* exception will be raised.

Return type *Transaction*

transfer_multiple(*destinations*, *metadata=None*, *allow_withdrawal=False*, *ttl=None*, *passphrase=None*)
Sends multiple transfers from the wallet. Returns the resulting transaction.

Parameters

- **destinations** – a list of *Address* and amount pairs `[(address, amount), ...]` or triples where the third element is a sequence of *AssetID* and quantity pairs
- **metadata** – metadata to be sent, as *Metadata* instance od `dict` mapping `int` keys to values of acceptable types
- **allow_withdrawal** – Allow withdrawing staking rewards to cover the transaction amount or fee.
- **ttl** – Time To Live in seconds. After TTL has lapsed the nodes give up on broadcasting the transaction. Leave *None* to use the default value.
- **passphrase** – the passphrase to the wallet. It takes precedence over *self.passphrase* and is discarded after use. If neither *self.passphrase* nor *passphrase* is set, a *MissingPassphrase* exception will be raised.

Return type *Transaction*

unstake(*passphrase=None*)
Cancels active stake delegation.

Parameters **passphrase** – the passphrase to the wallet. It takes precedence over *self.passphrase* and is discarded after use. If neither *self.passphrase* nor *passphrase* is set, a *MissingPassphrase* exception will be raised.

Return type *Transaction*

utxo_stats()
Returns UTXO statistics as a tuple of `(total_balance, histogram, scale)`.

class cardano.wallet.**WalletService**(*backend=None*)
Represents the service responsible for listing and retrieving the existing wallets or creating new ones.

Parameters **backend** – the backend used to handle the underlying service layer

create_wallet(*name*, *mnemonic*, *passphrase*, *mnemonic_2f=None*)
Creates/restores a wallet internally in the backend. Returns only ID as the backend may need some time to sync before being able to return full wallet data.

Parameters

- **name** – Name of the wallet
- **mnemonic** – The mnemonic seed
- **passphrase** – The wallet passphrase for spending operations (plain text string)
- **mnemonic_2f** – An optional passphrase used to encrypt the mnemonic sentence

Return type `str`

`wallet`(*wid*, *passphrase=None*)
Returns the wallet of given ID, connected to the backend and equipped with the passphrase if given.

Parameters

- **wid** – The wallet ID (hex string)
- **passphrase** – The wallet passphrase for spending operations (plain text string)

Return type *Wallet*

`wallets`()
Returns the list of all *Wallets* handled by the backend.

CHAPTER

THREE

ADDRESSES

At the moment the module doesn't have validation of Cardano addresses, other than checking if the prefix is correct. It recognizes the following:

- Shelley era `addr1` and `addr_test1`
- Byron era `Ae2` and `DdzFF`

Addresses are instances of *`Address`* class but you may use strings instead. Conversion and comparison methods are provided. No other functionality is available yet.

3.1 Retrieving wallet addresses

To get a list of addresses available in a wallet, do the following:

```
In [6]: wal.addresses()
Out[6]: [addr_
↪test1qr9ujxmsvdya6r4e9lxlu4n37svn52us7z8uzqdkhw8muqld56vd3zqzthdaweyrktfm3h5cz4je9h5j6s0f24pryswqzuzv
↪
        addr_
↪test1qpjqfw0xn8wp3rt9633ja6ua2nfmpx70qdn67cutc93p02hd56vd3zqzthdaweyrktfm3h5cz4je9h5j6s0f24pryswqvy6c
↪
        addr_
↪test1qqaaeru7xswhg9n9653ajpcryxl0334ryfp3kpuvd6aw0hhd56vd3zqzthdaweyrktfm3h5cz4je9h5j6s0f24pryswqukue
↪
        addr_
↪test1qqmv6m3mjwk8xfwc6mmxrah5fgrvvvtk0ncey84jcs4a798d56vd3zqzthdaweyrktfm3h5cz4je9h5j6s0f24pryswqzn97
↪
        addr_
↪test1qznf0k97a9wn50yy3aw6l2zfugknczj45gyfk2nykk49qy0d56vd3zqzthdaweyrktfm3h5cz4je9h5j6s0f24pryswq88su
↪
        addr_
↪test1qrkm3tgk74edkv60uqwedayw4kut0zgg5qgtm3epjvyxvt0d56vd3zqzthdaweyrktfm3h5cz4je9h5j6s0f24pryswqwu2s
↪
        addr_
↪test1qqd86dlwasc5kwe39m0qvu4v6krd24qek0g9pv9f2kq9x28d56vd3zqzthdaweyrktfm3h5cz4je9h5j6s0f24pryswqgepa
↪
        addr_
↪test1qrjvywxnrwv7ehx7f0enyta2n2lpjfk096df3mul9zr8vw8d56vd3zqzthdaweyrktfm3h5cz4je9h5j6s0f24pryswq737k
↪
        addr_
↪test1qzyhv3csdwlyuwt7kgjvkjpfrq0ap3d6lpm0ej88yf6zuj8d56vd3zqzthdaweyrktfm3h5cz4je9h5j6s0f24pryswqflta
↪
```

(continues on next page)

(continued from previous page)

```
        addr_
↪test1qpzwtycspdltafh34fedqayuh755uefuafvnveta6tt95z8d56vd3zqzthdaweyrktfm3h5cz4je9h5j6s0f24pryswqk2ems
↪
        addr_
↪test1qqfkfcpcdd0ll44qj3wasnl6vdyay3zur7lpay4p49pxjt8d56vd3zqzthdaweyrktfm3h5cz4je9h5j6s0f24pryswqghlc
↪
        addr_
↪test1qpd78m0427l4q62s305c0487gnqla07t33sdm9wenvgw23hd56vd3zqzthdaweyrktfm3h5cz4je9h5j6s0f24pryswqcrzd
↪
        addr_
↪test1qz3d7yap080hnkz8wjrmcng0euc5qamrl4s6daw0f8gwknld56vd3zqzthdaweyrktfm3h5cz4je9h5j6s0f24pryswq8eeq
↪
        addr_
↪test1qzunys8wvg5ssh2m6jrpvsckmjye30d7kzavrmq3gwkt270d56vd3zqzthdaweyrktfm3h5cz4je9h5j6s0f24pryswq85qd
↪
        addr_
↪test1qqdvqczhy05zjsspaeaay3a27xey0lm2lwevl27sgfy4y88d56vd3zqzthdaweyrktfm3h5cz4je9h5j6s0f24pryswq6luv
↪
        addr_
↪test1qqu76rcvmt5e86dyxv90dpch5adgafhdmy20hs36n07ds9hd56vd3zqzthdaweyrktfm3h5cz4je9h5j6s0f24pryswqgq0n
↪
        addr_
↪test1qp00p2y2yf9gkqul9w0jzfyju690flpefunjdl7z9cm2wdld56vd3zqzthdaweyrktfm3h5cz4je9h5j6s0f24pryswq0aw9
↪
        addr_
↪test1qzkh6wkkhgf57g5s6tqpt342fqezurx7tmapdw8q3mlud2hd56vd3zqzthdaweyrktfm3h5cz4je9h5j6s0f24pryswqnr9z
↪
        addr_
↪test1qr9y5q90w2e866gc5ehs5h2dqwzdd9maenpxf6wdj4c5238d56vd3zqzthdaweyrktfm3h5cz4je9h5j6s0f24pryswqspuw
↪
        addr_
↪test1qzhjjeqt42lc0g48mlljsjxlleu24q206vxgtd7fu3vrzyhd56vd3zqzthdaweyrktfm3h5cz4je9h5j6s0f24pryswqv5nv
```

Optionally, you may also retrieve info whether each of the addresses has been used. That means, it has received some funds. Please note the returned structure will be now a list of tuples `(address, used)`.

```
In [7]: wal.addresses(with_usage=True)
Out[7]: [(addr_
↪test1qr9ujxmsvdya6r4e9lxlu4n37svn52us7z8uzqdkhw8muqld56vd3zqzthdaweyrktfm3h5cz4je9h5j6s0f24pryswqzuzv
↪ True),
        (addr_
↪test1qp64xq7fsz9kvjwjy5tzfpetp2jmmhhk68kw066wqvyfgvhd56vd3zqzthdaweyrktfm3h5cz4je9h5j6s0f24pryswqj7ms
↪ True),
        (addr_
↪test1qpjqfw0xn8wp3rt9633ja6ua2nfmpx70qdn67cutc93p02hd56vd3zqzthdaweyrktfm3h5cz4je9h5j6s0f24pryswqvy6c
↪ False),
        (addr_
↪test1qqaaeru7xswhg9n9653ajpcryxl0334ryfp3kpuvd6aw0hhd56vd3zqzthdaweyrktfm3h5cz4je9h5j6s0f24pryswqukue
↪ False),
        (addr_
↪test1qqmv6m3mjwk8xfwc6mmxrah5fgrvvvtk0ncey84jcs4a798d56vd3zqzthdaweyrktfm3h5cz4je9h5j6s0f24pryswqzn97
↪ False),
```

(continues on next page)

(continued from previous page)

```
        (addr_
↪test1qznf0k97a9wn50yy3aw6l2zfugknczj45gyfk2nykk49qy0d56vd3zqzthdaweyrktfm3h5cz4je9h5j6s0f24pryswq88sus
↪ False),
        (addr_
↪test1qrkm3tgk74edkv60uqwedayw4kut0zgg5qgtm3epjvyxvt0d56vd3zqzthdaweyrktfm3h5cz4je9h5j6s0f24pryswqwu2s
↪ False),
        (addr_
↪test1qqd86dlwasc5kwe39m0qvu4v6krd24qek0g9pv9f2kq9x28d56vd3zqzthdaweyrktfm3h5cz4je9h5j6s0f24pryswqgepa
↪ False),
        (addr_
↪test1qrjvywxnrwv7ehx7f0enyta2n2lpjfk096df3mul9zr8vw8d56vd3zqzthdaweyrktfm3h5cz4je9h5j6s0f24pryswq737k
↪ False),
        (addr_
↪test1qzyhv3csdwlyuwt7kgjvkjpfrq0ap3d6lpm0ej88yf6zuj8d56vd3zqzthdaweyrktfm3h5cz4je9h5j6s0f24pryswqflta
↪ False),
        (addr_
↪test1qpzwtycspdltafh34fedqayuh755uefuafvnveta6tt95z8d56vd3zqzthdaweyrktfm3h5cz4je9h5j6s0f24pryswqk2em
↪ False),
        (addr_
↪test1qqfkfcpcdd0ll44qj3wasnl6vdyay3zur7lpay4p49pxjt8d56vd3zqzthdaweyrktfm3h5cz4je9h5j6s0f24pryswqghlc
↪ False),
        (addr_
↪test1qpd78m0427l4q62s305c0487gnqla07t33sdm9wenvgw23hd56vd3zqzthdaweyrktfm3h5cz4je9h5j6s0f24pryswqcrzd
↪ False),
        (addr_
↪test1qz3d7yap080hnkz8wjrmcng0euc5qamrl4s6daw0f8gwknld56vd3zqzthdaweyrktfm3h5cz4je9h5j6s0f24pryswq8eeq
↪ False),
        (addr_
↪test1qzunys8wvg5ssh2m6jrpvsckmjye30d7kzavrmq3gwkt270d56vd3zqzthdaweyrktfm3h5cz4je9h5j6s0f24pryswq85qd
↪ False),
        (addr_
↪test1qqdvqczhy05zjsspaeaay3a27xey0lm2lwevl27sgfy4y88d56vd3zqzthdaweyrktfm3h5cz4je9h5j6s0f24pryswq6luv
↪ False),
        (addr_
↪test1qqu76rcvmt5e86dyxv90dpch5adgafhdmy20hs36n07ds9hd56vd3zqzthdaweyrktfm3h5cz4je9h5j6s0f24pryswqgq0n
↪ False),
        (addr_
↪test1qp00p2y2yf9gkqul9w0jzfyju690flpefunjdl7z9cm2wdld56vd3zqzthdaweyrktfm3h5cz4je9h5j6s0f24pryswq0aw9
↪ False),
        (addr_
↪test1qzkh6wkkhgf57g5s6tqpt342fqezurx7tmapdw8q3mlud2hd56vd3zqzthdaweyrktfm3h5cz4je9h5j6s0f24pryswqnr9z
↪ False),
        (addr_
↪test1qr9y5q90w2e866gc5ehs5h2dqwzdd9maenpxf6wdj4c5238d56vd3zqzthdaweyrktfm3h5cz4je9h5j6s0f24pryswqspuw
↪ False),
        (addr_
↪test1qzhjjeqt42lc0g48mlljsjxlleu24q206vxgtd7fu3vrzyhd56vd3zqzthdaweyrktfm3h5cz4je9h5j6s0f24pryswqv5nv
↪ False),
        (addr_
↪test1qzhhm8em4pundp2ypcd36euplhe39pmuah290meu6l6gtqhd56vd3zqzthdaweyrktfm3h5cz4je9h5j6s0f24pryswqusvk
↪ False)]
```

3.2 API reference

class cardano.address.**Address**(*addr*, *wallet=None*)

Cardano base address class. Does no validation, it is up to child classes.

Compares with `str` and `bytes` objects.

Parameters

- **addr** – the address as `str` or `bytes` or `Address`
- **wallet** – the `Wallet` object if address belongs to

class cardano.address.**ByronAddress**(*addr*, *wallet=None*)

class cardano.address.**IcarusAddress**(*addr*, *wallet=None*)

class cardano.address.**ShelleyAddress**(*addr*, *wallet=None*)

CHAPTER

FOUR

LISTING TRANSACTIONS AND SPENDING FUNDS

From wallet perspective transactions can be generally grouped into incoming and outgoing. This is just a convenience, as almost every outgoing transaction has a change amount that goes back to the originating wallet.

4.1 Listing transactions

Retrieving the history of the wallet is pretty straightforward:

```
In [8]: txns = wal.transactions()

In [9]: txns[0].txid
Out[9]: '88633270f854eea5b2f35a863d748b294299deecf62ec9629ff08fca87fff45c'

In [10]: txns[0].amount_out
Out[10]: Decimal('1.000000')

In [11]: txns[0].fee
Out[11]: Decimal('0.168801')

In [12]: txns[1].txid
Out[12]: '0b048162778e29e98d833d948a3be7f18f9ce8693d7ee407c7d38b6ef2a5a264'
```

As you probably noticed, the amounts are given in ADA as Python `Decimal` type, which is perfect for monetary operations.

4.1.1 Narrowing down the query

In order to limit the number of results you may ask for transactions meeting special criteria.

The most important, perhaps, is the `txid` argument which accepts single IDs as well as sequences thereof. So, both

```
wal.transactions(txid="0b048162778e29e98d833d948a3be7f18f9ce8693d7ee407c7d38b6ef2a5a264")
```

as well as

```
wal.transactions(txid=[
    "0b048162778e29e98d833d948a3be7f18f9ce8693d7ee407c7d38b6ef2a5a264",
    "88633270f854eea5b2f35a863d748b294299deecf62ec9629ff08fca87fff45c"]
    )
```

are valid queries.

Blockchain position

The transaction filter accepts parameters filtering against the position in the ledger. `min_epoch`, `max_epoch`, `min_slot`, `max_slot`, `min_absolute_slot`, `max_absolute_slot`, `min_height` and `max_height` can be used and combined.

Ranges combining different criteria may be applied in the same call, e.g. to ask only for the first 10 slots of epoch 230 would be

```
wal.transactions(min_epoch=230, max_epoch=230, max_slot=10)
```

Because both epochs and slots are precisely defined periods of time, querying for them is like asking for quite precise timestamp of mining of the transaction's block. In contrast, asking for height considers the actual number of blocks since the genesis, as not all slots have been used to generate a block.

Mempool

Even though the mempool life part of Cardano transactions is usually very short, it is possible to ask for transactions not in ledger, as well as to exclude them from the results.

To include mempool, use `unconfirmed=True`. To include mined transactions, use `confirmed=True`. `False` values exclude these types of transactions from the results.

By default, `unconfirmed=False` and `confirmed=True` which means the default settings ask only for transactions in the ledger.

Filtering by address

Arguments `src_addr` and `dest_addr` filter for source and destination addresses, respectively. They can be used to ask for single or multiple addresses, just like `txid` described above.

Note: Please be aware that this kind of query is not very reliable. `cardano-wallet` is known to return incomplete input/output data, missing the address info.

4.2 Spending funds

In order to spend funds, you need to specify the destination address, amount (as `Decimal` or `int`) and provide the passphrase if you haven't done so when initializing the `Wallet` object.

```
In [17]: tx = wal.transfer(
    "addr_
↪test1qqr585tvlc7ylnqvz8pyqwauzrdu0mxag3m7q56grgmgu7sxu2hyfhlkwuxupa9d5085eunq2qywy7hvmvej456flknswgndm
↪",
    7,
    passphrase="xxx")

In [18]: tx.txid
Out[18]: 'a7a16a0653a6a397eb822ff8a3f610b5dabc82c5da2425fcc267f983f0edec88'

In [19]: tx.amount_in
```

(continues on next page)

(continued from previous page)

```
Out[19]: Decimal('0.000000')

In [20]: tx.amount_out
Out[20]: Decimal('7.000000')

In [21]: tx.fee
Out[21]: Decimal('0.168801')
```

Another useful function is `Wallet.transfer_multiple` which accepts more than one destination for a single transaction. It is useful for aggregating payouts and reducing fee costs. The difference from the previous method is that it accepts a sequence of `(address, amount)` pairs.

```
In [23]: tx = wallet.transfer_multiple(
        (
            (
                "addr_
↪test1qqr585tvlc7ylnqvz8pyqwauzrdu0mxag3m7q56grgmgu7sxu2hyfhlkwuxupa9d5085eunq2qywy7hvmvej456flknswgndu
↪",
                Decimal("1.234567"),
            ),
            (
                "addr_
↪test1qqd86dlwasc5kwe39m0qvu4v6krd24qek0g9pv9f2kq9x28d56vd3zqzthdaweyrktfm3h5cz4je9h5j6s0f24pryswqgepa
↪",
                Decimal("2.345678"),
            ),
        ),
        passphrase="xxx",
    )

In [24]: tx.txid
Out[24]: 'a7a16a0653a6a397eb822ff8a3f610b5dabc82c5da2425fcc267f983f0edec88'

In [25]: tx.amount_in
Out[25]: Decimal('0.000000')

In [26]: tx.amount_out
Out[26]: Decimal('3.580245')

In [27]: tx.fee
Out[27]: Decimal('0.168801')
```

Of course the list of destinations can have a single element. In fact, the `transfer()` method is just a shortcut for `transfer_multiple()` to make single payments easier.

4.2.1 Estimating fees

The `Wallet` object also offers method which estimates fee for transaction. The signature is similar to `transfer_multiple()`. It accepts a list of payments to be made and optionally the metadata, and returns a tuple of estimated minimum and maximum fee for the eventual transaction.

```
In [23]: f = wal.estimate_fee(
        (
            ("addr_
↪test1qqr585tvlc7ylnqvz8pyqwauzrdu0mxag3m7q56grgmgu7sxu2hyfhlkwuxupa9d5085eunq2qywy7hvmvej456flknswgndu
↪",
            Decimal("1.234567")),
            ("addr_
↪test1qqd86dlwasc5kwe39m0qvu4v6krd24qek0g9pv9f2kq9x28d56vd3zqzthdaweyrktfm3h5cz4je9h5j6s0f24pryswqgepa
↪",
            Decimal("2.345678")),
        ))

In [24]: f
Out[24]: (Decimal('0.174785'), Decimal('0.180989'))
```

Note: Don't forget to include metadata when estimating fees. They are based on the transaction size and additional data changes that significantly.

4.3 Metadata

Since the Shelley era, Cardano allows for adding metadata to transactions. Metadata is a mapping where keys are integers and values belong to a short list of supported data types. Description of the structure is beyond the scope of this documentation, however you may read this description or another one which includes a good test example.

4.3.1 Lists and dicts as map keys

While Cardano supports `map` objects that use another `map` or `list` as key element, this feature cannot be supported by the Python module directly. The reason is that data on blockchain is immutable (cannot be modified) while the corresponding Python objects (`dict` and `list`) are mutable, which disqualifies them as `dict` keys due to unstable hash value.

For that reason, substitutions have been introduced when following types of variables are used as keys:

- `list`: the key on Python side is `tuple`,
- `dict`: will be converted to `ImmutableDict`

4.3.2 Storing and retrieving metadata

Metadata can be passed to `Wallet.transfer()` and `Wallet.transfer_multiple()` methods as `dict` or `Metadata` instance. It will be instantly available in the `.metadata` attribute of the resulting `Transaction` object.

```
In [23]: tx = wal.transfer(
    "addr_
↪test1qqr585tvlc7ylnqvz8pyqwauzrdu0mxag3m7q56grgmgu7sxu2hyfhlkwuxupa9d5085eunq2qywy7hvmvej456flknswgndu
↪",
    7,
    metadata={1: "first value", 23: "next value"},
    passphrase="xxx")

In [24]: tx.metadata
Out[24]: {1: 'first value', 23: 'next value'}
```

Metadata can be initialized separately by passing a list of `(key, value)` pairs.

```
In [25]: m = Metadata(((1, "first value"), (23, "next value")))

In [26]: m
Out[26]: {1: 'first value', 23: 'next value'}
```

Such instance can be also passed as `metadata` parameter to the transfer methods.

4.4 API reference

4.4.1 Transactions

class cardano.transaction.**Input**(*iid=None*, *address=None*, *amount=None*, *assets=None*)

Represents a *Transaction* input.

Parameters

- **iid** (str hex) – the input ID
- **address** (*cardano.address.Address*) – the origin address
- **amount** (Decimal) – the amount in ADA
- **assets** (list) – a sequence of *AssetID* quantity pairs

class cardano.transaction.**Output**(*address=None*, *amount=None*, *assets=None*)

Represents a *Transaction* output.

Parameters

- **address** (*cardano.address.Address*) – the destination address
- **amount** (Decimal) – the amount in ADA
- **assets** (list) – a sequence of *AssetID* quantity pairs

class cardano.transaction.**Transaction**(*txid=None*, ***kwargs*)

Represents a Cardano transaction.

Parameters

- **txid** – the ID of the transaction

- **fee** – fee amount in ADA
- **inputs** – a sequence of *Input* objects
- **outputs** – a sequence of *Output* objects
- **local_inputs** – a sequence of *Input* objects that originate from local wallet
- **local_outputs** – a sequence of *Output* objects that are destined to local wallet
- **withdrawals** – a sequence of (Decimal, str) pairs of amounts and stake addresses
- **metadata** – an instance of *Metadata*

4.4.2 Numbers

A submodule with helpers useful for unit conversion. The idea is to represent amounts in ADA as Decimal type with 6 places of precision. For low-level backends, however, it's easier to use int of Lovelaces.

Also, float arguments are accepted but will issue a RuntimeWarning as it is a **very bad idea** to use floating-point numbers for monetary data.

cardano.numbers.**as_ada**(*amount*)
Return the amount rounded to maximal ADA precision.

Parameters **amount** (*Decimal*, *int*) – the amount to be sanitized

Return type Decimal with 6 decimal places precision

cardano.numbers.**from_lovelaces**(*amount*)
Convert Lovelaces to ADA.

Parameters **amount** (*int*) – the amount of Lovelaces

Return type Decimal

cardano.numbers.**to_lovelaces**(*amount*)
Convert ADA to Lovelaces.

Parameters **amount** (*Decimal*, *int*) – the amount of ADA

Return type int

4.4.3 Metadata

A class representing Cardano transaction metadata. Inherits from dict and offers both validation and serialization of the data.

class cardano.metadata.**ImmutableDict**
A flavor of dict with all mutating methods blocked and hash generation added. It can be used as mapping keys.

clear() → None. Remove all items from D.

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised

popitem() → (k, v), remove and return some (key, value) pair as a
2-tuple; but raise KeyError if D is empty.

update([*E*], **F) → None. Update D from dict/iterable E and F.
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

class cardano.metadata.**Metadata**(**args*, ***kwargs*)

Represents Cardano transaction metadata. Inherits from `dict` but passes all keys and values through validity check.

Parameters **mapping** – a sequence of (key, value) pairs

static deserialize(*txdata*)

Deserializes transaction metadata `dict` and returns *Metadata* instance.

Parameters **txdata** – the transaction data

serialize()

Returns serialized form of the metadata, which can be passed to the transaction.

static serialize_value(*val*)

Serializes Python value to an object that can be passed to transaction as a metadata value. The returned object is a mapping which contains both the type name and the value.

Raises `RuntimeError` when a value of unrecognized type has been passed.

static validate_key(*key*)

Checks if the key is allowed, i.e. is an `int` and within the allowed range.

Raises `KeyError` otherwise.

static validate_value(*val*)

Checks if the value is allowed, i.e. is one of `int`, `str`, `bytes`, `bytearray`, `list` or `dict`.

Raises `TypeError` otherwise. Also raises `ValueError` if the value is of proper type but exceeds range limit for Cardano metadata.

CHAPTER

FIVE

WORKING WITH NATIVE ASSETS

With Mary era Cardano introduced native assets to the network. Unlike other popular platforms, Cardano doesn't need smart contracts to handle them. The assets are indivisible (amounts are integers) and their handling is somewhat different than of ADA.

5.1 Asset IDs

Since different native assets may bear the same name, the actual identifier of an asset consists of `asset_name` and `policy_id`. They are grouped together into *cardano.simpletypes.AssetID*, a class which supports equality operator.

5.2 Balances

The wallet has `.assets()` method which returns a dict where keys are *cardano.simpletypes.AssetID* and values are *cardano.simpletypes.Balance* objects. At the moment the balances have always `None` as reward and total equal to available but that may perhaps change in the future.

```
In [9]: wal.assets()
Out[9]: {6c6f766164616e6674:0c306361512844fbdb83294f278937c04af6e56ab1d94d2dd187d725:␣
↪Balance(total=1, available=1, reward=None),␣
↪6c6f766164616e6674:0f5e9e9143f4eb0317584aa295d0d2dc9741edfdbbe1af64f241aa32:␣
↪Balance(total=1, available=1, reward=None)}
```

5.3 Sending assets

Transfer of assets can be specified by additional keyword to the `Wallet.transfer()` function or third element of `destinations` item passed to `Wallet.transfer_multiple()`. An example of sending 2.0 ADA along with a single native token is:

```
In [10]: wal.transfer(
         "addr_
↪test1qqpwa4lv202c9q4fag5kepr0jjnreq8yxrjgau7u4ulppa9c69u4ed55s8p7nuef3z65fkjjxcslwdu3h75zl7zeuzgqv3l7
↪",
         2,
         assets=[
             (
```

(continues on next page)

(continued from previous page)

```
                AssetID(
                    "7461786174696f6e206973207468656674",
                    "6b8d07d69639e9413aa637a1a815a7123c69c86abbafb66dbfdb1aa7"
                    ),
                1)
        ])
```

CHAPTER

SIX

STAKING

The module allows for staking ADA (also called delegation), withdrawing the stake and retrieving information about pools. This part of the functionality relies heavily on custom data structures, so you may also *check their descriptions*.

6.1 Querying pools

To query for a list of staking pools, `Wallet.stake_pools()` may be used. It accepts the amount of the stake as optional argument, otherwise it uses wallet's current total balance.

The result is a list of pools, represented by *`StakePoolInfo`* and sorted by expected long-term interest, from highest to lowest.

```
In [30]: pools = wallet.stake_pools()

In [31]: pools[0]
Out[31]: StakePoolInfo(id='pool1tzmx7k40sm8kheam3pr2d4yexrp3jmv8l50suj6crnvn6dc2429',␣
↪status=<StakePoolStatus.ACTIVE: 1>, ticker=None, name=None, description=None,␣
↪homepage=None, rewards=StakeRewardMetrics(expected=Decimal('0.182832'), stake=Decimal(
↪'1051.689055')), cost=Decimal('340.000000'), margin=Decimal('0.035'), pledge=Decimal(
↪'54000000.000000'), relative_stake=Decimal('0.0014'), saturation=Decimal('0.
↪7001171530343129'), produced_blocks=1091, retirement=None)
```

You may query pools regardless of whether the wallet is currently delegating or not.

6.2 Staking status

The wallet might be in one of two states: delegating or not delegating. Additionally, once delegation or withdrawal have been scheduled, they would also appear in the list of planned future operations.

```
In [32]: wallet.staking_status()
Out[32]: (StakingStatus(delegating=True, target_id=
↪'pool1tzmx7k40sm8kheam3pr2d4yexrp3jmv8l50suj6crnvn6dc2429', changes_at=None), [])
```

The second element of the tuple returned by `Wallet.staking_status()` is a list of scheduled future staking status changes.

6.3 Staking

Once decided which stake pool to use, you may delegate the wallet's balance there. The first argument must be either pool's ID or one of the *StakePoolInfo* objects returned from pool query described above.

If successful, the result will be the delegation transaction.

```
In [33]: tx = wallet.stake("pool1xqh4kl5gzn4av7uf32lxas5k8tsfgvhy3hlnrg0fdp98q42jswr")

In [34]: tx.amount_out
Out[34]: Decimal('2.000000')
```

6.4 Unstaking

Cancelling an ongoing delegation is pretty straightforward. Just use the `Wallet.unstake()` method, providing a passphrase and you will get the unstaking transaction as the result.

However, if you have a positive reward balance in the wallet, it needs to be withdrawn first. You may do it by calling the `Wallet.transfer()` method for an amount higher than the accumulated reward and directing it to your first unused local address, for example:

```
In [33]: wallet.balance()
Out[33]: Balance(total=Decimal('1050.770234'), available=Decimal('1050.520254'),␣
↪reward=Decimal('0.249980'))

In [34]: wtx = wallet.transfer(wallet.first_unused_address(), Decimal(1), allow_
↪withdrawal=True)

In [35]: wtx.withdrawals
Out[35]: [(Decimal('0.249980'), 'stake_
↪test1urk6dxxc3qp9mk7hvjpm95acm6vp2evjm6fdg8542s3jg8qtsgmvf')]

In [36]: utx = wallet.unstake()

In [37]: wallet.staking_status()
Out[37]: (StakingStatus(delegating=True, target_id=
↪'pool1tzmx7k40sm8kheam3pr2d4yexrp3jmv8l50suj6crnvn6dc2429', changes_at=None),␣
↪[StakingStatus(delegating=False, target_id=None, changes_at=Epoch(number=134,␣
↪starts=datetime.datetime(2021, 5, 24, 20, 20, 16, tzinfo=tzutc()))])
```

CHAPTER

SEVEN

UTXO STATS

The *Wallet* has a `utxo_stats()` method which returns a histogram of UTXO (Unspent Transaction Output) statistics.

The result consists of three elements: total balance, histogram, scale.

The histogram part is a list of `(threshold, number)` pairs where the number describes how many UTXOs are available between the given threshold and the lower one.

The scale so far is always `"log10"`.

```
In [40]: total, dist, scale = wallet.utxo_stats()

In [41]: total
Out[41]: Decimal('1052.422864')

In [42]: scale
Out[42]: 'log10'

In [43]: print("\n".join(["{:18.6f}: {:4d}".format(*d) for d in dist.items()]))
          0.000010:    0
          0.000100:    0
          0.001000:    0
          0.010000:    0
          0.100000:    0
          1.000000:    1
         10.000000:   16
        100.000000:    0
       1000.000000:    2
      10000.000000:    0
     100000.000000:    0
    1000000.000000:    0
   10000000.000000:    0
  100000000.000000:    0
 1000000000.000000:    0
10000000000.000000:    0
45000000000.000000:    0
```

CHAPTER

EIGHT

CONSTANTS AND SIMPLE TYPES

These submodules contain constants and simple types used to represent Cardano data structures that don't need any internal functionality.

8.1 API reference

class cardano.consts.**Era**(*value*)
Represents Cardano era, a distinct phase of platform development.

> **Warning: Do NOT use the integer values directly.** There are new Eras being introduced, as for example Allegra and Mary were inserted as stepping stones into full Goguen. However, you may use comparison operators between them, to check which was earlier or later than the other one.

class cardano.simpletypes.**AssetID**(*asset_name*, *policy_id*)
Represents the ID of a native Cardano asset. It consists of asset name and policy ID. It renders as string representation of `asset_name:policy_id`.

The `asset_name` is always kept encoded as hexadecimal string and must be passed to the constructor as such.

The `.name_bytes` property is a `bytes` decoded representation of the hex. Because Cardano allows full ASCII set to be used in asset names, some of them are not safe to be displayed directly.

class cardano.simpletypes.**Balance**(*total*, *available*, *reward*)
Represents a balance of asset, including total, principal and reward

property available
The principal, i.e. the total minus staking rewards

property reward
The staking rewards (interest)

property total
The total balance

class cardano.simpletypes.**BlockPosition**(*epoch*, *slot*, *absolute_slot*, *height*)
Represents block's position within the blockchain

property absolute_slot
Absolute slot number

property epoch
Epoch number

property height
Block number (height of the chain) [optional]

property slot
Slot number

class cardano.simpletypes.**Epoch**(*number*, *starts*)

property number
Alias for field number 0

property starts
Alias for field number 1

class cardano.simpletypes.**StakePoolInfo**(*id*, *status*, *ticker*, *name*, *description*, *homepage*, *rewards*, *cost*, *margin*, *pledge*, *relative_stake*, *saturation*, *produced_blocks*, *retirement*)

Stores stake pool data

property cost
Fixed pool running cost in ADA

property description
Description

property homepage
Homepage URL

property id
Unique ID

property margin
Operator's margin on the total reward before splitting it among stakeholders (as `Decimal` fraction)

property name
Name

property pledge
Minimal stake amount that the pool is willing to honor

property produced_blocks
Number of blocks produced by a given stake pool in its lifetime.

property relative_stake
The live pool stake relative to the total stake

property retirement
The *Epoch* in which the pool retires

property rewards
Alias for field number 6

property saturation
Saturation-level of the pool based on the desired number of pools aimed by the network. A value above 1 indicates that the pool is saturated.

property status
Status, one of *StakePoolStatus* enum

property ticker
3-5 chars long ticker

class cardano.simpletypes.**StakePoolStatus**(*value*)

Represents stake pool status

class cardano.simpletypes.**StakeRewardMetrics**(*expected*, *stake*)
Represents stake pool reward metrics

property expected
Expected rewards at the end of an epoch, in ADA

property stake
Staked amount against which rewards were calculated, in ADA

class cardano.simpletypes.**StakingStatus**(*delegating*, *target_id*, *changes_at*)
Wallet's staking status

property changes_at
Epoch since which the change comes live

property delegating
Whether the wallet is actively delegating

property target_id
The ID of the pool the wallet is delegating to

CHAPTER

NINE

EXCEPTIONS

A collection of Cardano exceptions.

9.1 API reference

exception cardano.exceptions.**AlreadyWithdrawing**
Raised whan another withdrawal attempt is being made while one is already pending.

exception cardano.exceptions.**BackendException**
The base exception for backend errors.

exception cardano.exceptions.**CannotCoverFee**

exception cardano.exceptions.**CardanoException**
The base exception for cardano-python module.

exception cardano.exceptions.**MissingPassphrase**
Raised when the wallet is missing a required passphrase.

exception cardano.exceptions.**NonNullRewards**
Raised when trying to cancel stake without withdrawing rewards first.

exception cardano.exceptions.**NotEnoughMoney**
Raised when the balance is too low.

exception cardano.exceptions.**PoolAlreadyJoined**
Raised when trying to double-stake.

exception cardano.exceptions.**StakingException**
Base error when delegating, withdrawing or cancelling stake.

exception cardano.exceptions.**TransactionException**
Base for errors with constructing or handling transactions.

exception cardano.exceptions.**UTXOTooSmall**
Raised when the resulting UTXO with assets has too small ADA amount.

exception cardano.exceptions.**WalletAlreadyExists**
Raised when a duplicate wallet is requested to be created at the service.

exception cardano.exceptions.**WalletException**
The base exception for wallet errors.

exception cardano.exceptions.**WalletServiceException**
The base exception for wallet service errors.

exception cardano.exceptions.**WrongPassphrase**
Raised when the provided passphrase doesn't match the wallet's.

CHAPTER

TEN

THE WALLET REST BACKEND (FROM *CARDANO-WALLET*)

10.1 API reference

exception cardano.backends.walletrest.exceptions.**BadRequest**(**args*, ***kwargs*)
Raised when the underlying REST API returns HTTP code 400.

exception cardano.backends.walletrest.exceptions.**CreatedInvalidTransaction**(**args*, ***kwargs*)

exception cardano.backends.walletrest.exceptions.**NotFound**(**args*, ***kwargs*)
Raised when the underlying REST API returns HTTP code 404.

exception cardano.backends.walletrest.exceptions.**NotSupported**(**args*, ***kwargs*)
Raised when wallet doesn't provide the requested feature.

exception cardano.backends.walletrest.exceptions.**RESTServerError**(**args*, ***kwargs*)
Raised when the underlying REST API returns HTTP code 403 or 500 and the error cannot be handled.

exception cardano.backends.walletrest.exceptions.**WalletRESTException**(**args*, ***kwargs*)

CHAPTER

ELEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

C

INDEX

N

O

P

R

S

T

U

V

W