Transaction Module

This module provides the preparatory component that is required for sending a transaction.

with_message(msgs)

This function add one or more messages to

The message can be any message as listed in Oracle Modules or Cosmos Based Messages . Please note that our message should be imported from the generated protobuf files .

Parameter

- msgs
-
- : Messages to be included into the transaction.

Return

-

with_sender(client, sender)

This function setaccount_num andsequence from with the address fromsender . must have at least 1 message added before callingwith_sender()

Parameter

- client
-
- : Client used to setaccount_num
- andsequence
- by callingget_address()
- .
- sender
-
- : Address of the sender.

Return

-

Exception

Type Description EmptyMsgError Message is empty, please use with_messages at least 1 message NotFoundError Account doesn't exist

with_account_num(account_num)

This function sets the account number in.

Parameter

- account_num
-

Return

-

with_sequence(sequence)

This function sets the sequence number in.

Parameter

- sequence
-

Return

-

with_chain_id(chain_id)

This function sets the chain ID in.

Parameter

- chain_id
-

Return

-

with_fee(fee)

This function sets the fee by using the given fee and gas limit.

Parameter

- fee
-

Return

-

with_gas(gas)

This function sets the gas limit in.

Parameter

- gas
-

Return

-

with_memo(memo)

This function sets the memo in.

Parameter

- memo
-
- : Maximum length of memo is 256.

Return

-

Exception

Type Description ValueTooLargeError Memo is too large

get_sign_doc(public_key)

This function returns a sign data from.

Parameter

- public_key
-
- , default = None: Public key.

Return

-

Exception

Type Description EmptyMsgError message is empty UndefinedError account_num should be defined UndefinedError sequence should be defined UndefinedError chain_id should be defined

get_tx_data(signature, public_key)

This function returns a transaction that need to be sent.

Parameter

- signature
-
- : Signature from sign from get_sign_doc
- public_key
- <PublicKey
- , default = None: Public key

Return

-

Example use case

Note: Get the here

import os

from pyband . client import Client from pyband . transaction import Transaction from pyband . wallet import PrivateKey

from pyband . proto . cosmos . base . v1beta1 . coin_pb2 import Coin from pyband . proto . oracle . v1 . tx_pb2 import MsgRequestData

grpc_url

"" c = Client (grpc_url)

MNEMONIC

os . getenv ("MNEMONIC") private_key = PrivateKey . from_mnemonic (MNEMONIC) public_key = private_key . to_public_key ( ) sender_addr = public_key . to_address ( ) sender = sender_addr . to_acc_bech32 ( )

request_msg

MsgRequestData (oracle_script_id = 37 , calldata = bytes . fromhex ("0000000200000003425443000000034554480000000000000064") , ask_count = 4 , min_count = 3 , client_id = "BandProtocol" , fee_limit = [Coin (amount = "100" , denom = "uband")] , prepare_gas = 50000 , execute_gas = 200000 , sender = sender ,)

account

c . get_account (sender) account_num = account . account_number sequence = account . sequence

fee

[Coin (amount = "0" , denom = "uband")] chain_id = c . get_chain_id ( )

txn

(Transaction ( ) . with_messages (request_msg) . with_sequence (sequence) . with_account_num (account_num) . with_chain_id (chain_id) . with_gas (2000000) . with_fee (fee) . with_memo (""))

sign_doc

txn . get_sign_doc (public_key) signature = private_key . sign (sign_doc . SerializeToString ( )) tx_raw_bytes = txn . get_tx_data (signature , public_key)

print (tx_raw_bytes . hex ( ))

Result

```
0a93010a90010a192f6f7261636c652e76312e4d7367526571756573744461746112730825121a000000020000000342544300000003455448000000000000000064180420032a0c42616e6450726f746f636
```