algosdk

Aug 03, 2020

Contents

A python library for interacting with the Algorand network.

CHAPTER 1

Installation

Run `$ pip3 install py-algorand-sdk` to install the package.

Alternatively, choose a distribution file, and run `$ pip3 install [file name]`.

CHAPTER 2

SDK Development

Run tests with `make docker-test`

CHAPTER 3

Quick start

Here's a simple example you can run without a node.

```
from algosdk import account, encoding

# generate an account
private_key, address = account.generate_account()
print("Private key:", private_key)
print("Address:", address)

# check if the address is valid
if encoding.is_valid_address(address):
    print("The address is valid!")
else:
    print("The address is invalid.")
```

CHAPTER 4

Node setup

Follow the instructions in Algorand's developer resources to install a node on your computer.

CHAPTER 5

Running example.py

Before running example.py, start kmd:

```
$ ./goal kmd start -d [data directory]
```

Next, create a wallet and an account:

```
$ ./goal wallet new [wallet name] -d [data directory]
```

```
$ ./goal account new -d [data directory] -w [wallet name]
```

Visit the Algorand dispenser and enter the account address to fund your account.

Next, in params.py, either update the tokens and addresses, or provide a path to the data directory.

You're now ready to run example.py!

CHAPTER 6

More examples

6.1 using the Wallet class

Instead of always having to keep track of handles, IDs, and passwords for wallets, create a Wallet object to manage everything for you.

```
import params
from algosdk import kmd
from algosdk.wallet import Wallet

# create a kmd client
kcl = kmd.KMDClient(params.kmd_token, params.kmd_address)

# create a wallet object
wallet = Wallet("wallet_name", "wallet_password", kcl)

# get wallet information
info = wallet.info()
print("Wallet name:", info["wallet"]["name"])

# create an account
address = wallet.generate_key()
print("New account:", address)

# delete the account
delete = wallet.delete_key(address)
print("Account deleted:", delete)
```

6.2 backing up a wallet with mnemonic

```
import params
from algosdk import kmd, mnemonic
from algosdk.wallet import Wallet

# create a kmd client
kcl = kmd.KMDClient(params.kmd_token, params.kmd_address)

# create a wallet object
wallet = Wallet("wallet_name", "wallet_password", kcl)

# get the wallet's master derivation key
mdk = wallet.export_master_derivation_key()
print("Master Derivation Key:", mdk)

# get the backup phrase
backup = mnemonic.from_master_derivation_key(mdk)
print("Wallet backup phrase:", backup)
```

You can also back up accounts using mnemonic.from_private_key().

6.3 recovering a wallet using a backup phrase

```
import params
from algosdk import kmd, mnemonic

# get the master derivation key from the mnemonic
backup = "such chapter crane ugly uncover fun kitten duty culture giant skirt reunion␣
↪pizza pill web monster upon dolphin aunt close marble dune kangaroo ability merit"
mdk = mnemonic.to_master_derivation_key(backup)

# create a kmd client
kcl = kmd.KMDClient(params.kmd_token, params.kmd_address)

# recover the wallet by passing mdk when creating a wallet
kcl.create_wallet("wallet_name", "wallet_password", master_deriv_key=mdk)
```

You can also recover accounts using mnemonic.to_private_key().

6.4 writing transactions to file

If you don't want to send your transactions now, you can write them to file. This works with both signed and unsigned transactions.

```
import params
from algosdk import algod, kmd
from algosdk.future import transaction

sender = "sender_address"
receiver = "receiver_address"

# create an algod and kmd client
acl = algod.AlgodClient(params.algod_token, params.algod_address)
kcl = kmd.KMDClient(params.kmd_token, params.kmd_address)
```

(continues on next page)

(continued from previous page)

```
# get suggested parameters
sp = acl.suggested_params()

# create a transaction
amount = 10000
txn = transaction.PaymentTxn(sender, sp, receiver, amount)

# write to file
txns = [txn]
transaction.write_to_file([txn], "pathtofile.tx")
```

We can also read transactions after writing them to file.

```
# read from file
read_txns = transaction.retrieve_from_file("pathtofile.tx")
```

6.5 manipulating multisig transactions

```
import params
from algosdk import account, algod, encoding
from algosdk.future import transaction

acl = algod.AlgodClient(params.algod_token, params.algod_address)

# generate three accounts
private_key_1, account_1 = account.generate_account()
private_key_2, account_2 = account.generate_account()
private_key_3, account_3 = account.generate_account()

# create a multisig account
version = 1  # multisig version
threshold = 2  # how many signatures are necessary
msig = transaction.Multisig(version, threshold, [account_1, account_2])

# get suggested parameters
sp = acl.suggested_params()

# create a transaction
sender = msig.address()
amount = 10000
txn = transaction.PaymentTxn(sender, sp, account_3, amount)

# create a SignedTransaction object
mtx = transaction.MultisigTransaction(txn, msig)

# sign the transaction
mtx.sign(private_key_1)
mtx.sign(private_key_2)

# print encoded transaction
print(encoding.msgpack_encode(mtx))
```

6.6 working with NoteField

We can put things in the "note" field of a transaction; here's an example with an auction bid. Note that you can put any bytes you want in the "note" field; you don't have to use the NoteField object.

```
from algosdk import algod, mnemonic, account
from algosdk.future import transaction

passphrase = "teach chat health avocado broken avocado trick adapt parade witness␣
↪damp gift behave harbor maze truth figure below scatter taste slow sustain aspect␣
↪absorb nuclear"

acl = algod.AlgodClient("API-TOKEN", "API-Address")

# convert passphrase to secret key
sk = mnemonic.to_private_key(passphrase)

# get suggested parameters
sp = acl.suggested_params()

# Set other parameters
amount = 100000
note = "Some Text".encode()
receiver = "receiver Algorand Address"

# create the transaction
txn = transaction.PaymentTxn(account.address_from_private_key(sk), sp, receiver,␣
↪amount, note=note)

# sign it
stx = txn.sign(sk)

# send it
txid = acl.send_transaction(stx)
```

We can also get the NoteField object back from its bytes:

```
# decode notefield
decoded = encoding.msgpack_decode(base64.b64encode(note_field_bytes))
print(decoded.dictify())
```

6.7 working with transaction group

```
import params
from algosdk import algod, kmd
from algosdk.future import transaction

private_key_sender, sender = account.generate_account()
private_key_receiver, receiver = account.generate_account()

# create an algod and kmd client
acl = algod.AlgodClient(params.algod_token, params.algod_address)
kcl = kmd.KMDClient(params.kmd_token, params.kmd_address)
```

(continues on next page)

(continued from previous page)

```
# get suggested parameters
sp = acl.suggested_params()

# create a transaction
amount = 10000
txn1 = transaction.PaymentTxn(sender, sp, receiver, amount)
txn2 = transaction.PaymentTxn(receiver, sp, sender, amount)

# get group id and assign it to transactions
gid = transaction.calculate_group_id([txn1, txn2])
txn1.transaction.group = gid
txn2.transaction.group = gid

# sign transactions
stxn1 = txn1.sign(private_key_sender)
stxn2 = txn2.sign(private_key_receiver)

# send them over network
acl.send_transactions([stxn1, stxn2])
```

6.8 working with logic sig

Example below creates a LogicSig transaction signed by a program that never approves the transfer.

```
import params
from algosdk import algod
from algosdk.future import transaction

program = b"\x01\x20\x01\x00\x22"  # int 0
lsig = transaction.LogicSig(program)
sender = lsig.address()

# create an algod client
acl = algod.AlgodClient(params.algod_token, params.algod_address)

# get suggested parameters
sp = acl.suggested_params()

# create a transaction
amount = 10000
txn = transaction.PaymentTxn(sender, sp, receiver, amount)

# note, transaction is signed by logic only (no delegation)
# that means sender address must match to program hash
lstx = transaction.LogicSigTransaction(txn, lsig)
assert lstx.verify()

# send them over network
acl.send_transaction(lstx)
```

6.9 working with assets

Assets can be managed by sending three types of transactions: AssetConfigTxn, AssetFreezeTxn, and AssetTransferTxn. Shown below are examples of how to use these transactions.

6.9.1 creating an asset

```
from algosdk import account
from algosdk.future import transaction

private_key, address = account.generate_account() # creator
_, freeze = account.generate_account() # account that can freeze other accounts for␣
↪this asset
_, manager = account.generate_account() # account able to update asset configuration
_, clawback = account.generate_account() # account allowed to take this asset from␣
↪any other account
_, reserve = account.generate_account() # account that holds reserves for this asset

fee_per_byte = 10
first_valid_round = 1000
last_valid_round = 2000
genesis_hash = "SGO1GKSzyE7IEPItTxCByw9x8FmnrCDexi9/cOUJOiI="

total = 100 # how many of this asset there will be
assetname = "assetname"
unitname = "unitname"
url = "website"
metadata = bytes("fACPO4nRgO55j1ndAK3W6Sgc4APkcyFh", "ascii") # should be a 32-byte␣
↪hash
default_frozen = False # whether accounts should be frozen by default

# create the asset creation transaction
sp = transaction.SuggestedParams(fee_per_byte, first_valid_round, last_valid_round,␣
↪genesis_hash)
txn = transaction.AssetConfigTxn(address, sp, total=total, manager=manager,
            reserve=reserve, freeze=freeze, clawback=clawback,
            unit_name=unitname, asset_name=assetname, url=url,
            metadata_hash=metadata, default_frozen=default_frozen)

# sign the transaction
signed_txn = txn.sign(private_key)
```

6.9.2 updating asset configuration

This transaction must be sent from the manager's account.

```
from algosdk import account
from algosdk.future import transaction

manager_private_key = "manager private key"
manager_address = "manager address"
_, new_freeze = account.generate_account() # account that can freeze other accounts␣
↪for this asset
_, new_manager = account.generate_account() # account able to update asset␣
↪configuration
```

(continues on next page)

(continued from previous page)

```
_, new_clawback = account.generate_account() # account allowed to take this asset␣
↪from any other account
_, new_reserve = account.generate_account() # account that holds reserves for this␣
↪asset

fee_per_byte = 10
first_valid_round = 1000
last_valid_round = 2000
genesis_hash = "SGO1GKSzyE7IEPItTxCByw9x8FmnrCDexi9/cOUJOiI="

index = 1234 # identifying index of the asset

# create the asset config transaction
sp = transaction.SuggestedParams(fee_per_byte, first_valid_round, last_valid_round,␣
↪genesis_hash)
txn = transaction.AssetConfigTxn(manager_address, sp, manager=new_manager,␣
↪reserve=new_reserve,
            freeze=new_freeze, clawback=new_clawback, index=index)

# sign the transaction
signed_txn = txn.sign(manager_private_key)
```

6.9.3 destroying an asset

This transaction must be sent from the creator's account.

```
from algosdk import account
from algosdk.future import transaction

creator_private_key = "creator private key"
creator_address = "creator address"

fee_per_byte = 10
first_valid_round = 1000
last_valid_round = 2000
genesis_hash = "SGO1GKSzyE7IEPItTxCByw9x8FmnrCDexi9/cOUJOiI="

index = 1234 # identifying index of the asset

# create the asset destroy transaction
sp = transaction.SuggestedParams(fee_per_byte, first_valid_round, last_valid_round,␣
↪genesis_hash)
txn = transaction.AssetConfigTxn(creator_address, sp, index=index, strict_empty_
↪address_check=False)

# sign the transaction
signed_txn = txn.sign(creator_private_key)
```

6.9.4 freezing or unfreezing an account

This transaction must be sent from the account specified as the freeze manager for the asset.

```
from algosdk import account
from algosdk.future import transaction

freeze_private_key = "freeze private key"
freeze_address = "freeze address"

fee_per_byte = 10
first_valid_round = 1000
last_valid_round = 2000
genesis_hash = "SGO1GKSzyE7IEPItTxCByw9x8FmnrCDexi9/cOUJOiI="
freeze_target = "address to be frozen or unfrozen"

index = 1234 # identifying index of the asset

# create the asset freeze transaction
sp = transaction.SuggestedParams(fee_per_byte, first_valid_round, last_valid_round,␣
↪genesis_hash)
txn = transaction.AssetFreezeTxn(freeze_address, sp, index=index, target=freeze_
↪target,
            new_freeze_state=True)

# sign the transaction
signed_txn = txn.sign(freeze_private_key)
```

6.9.5 sending assets

```
from algosdk import account
from algosdk.future import transaction

sender_private_key = "freeze private key"
sender_address = "freeze address"

fee_per_byte = 10
first_valid_round = 1000
last_valid_round = 2000
genesis_hash = "SGO1GKSzyE7IEPItTxCByw9x8FmnrCDexi9/cOUJOiI="
close_assets_to = "account to close assets to"
receiver = "account to receive assets"
amount = 100 # amount of assets to transfer

index = 1234 # identifying index of the asset

# create the asset transfer transaction
sp = transaction.SuggestedParams(fee_per_byte, first_valid_round, last_valid_round,␣
↪genesis_hash)
txn = transaction.AssetTransferTxn(sender_address, sp,
                receiver, amount, index, close_assets_to)

# sign the transaction
signed_txn = txn.sign(sender_private_key)
```

6.9.6 accepting assets

```
from algosdk import account
from algosdk.future import transaction

private_key = "freeze private key"
address = "freeze address"

fee_per_byte = 10
first_valid_round = 1000
last_valid_round = 2000
genesis_hash = "SGO1GKSzyE7IEPItTxCByw9x8FmnrCDexi9/cOUJOiI="
receiver = address # to start accepting assets, set receiver to sender
amount = 0 # to start accepting assets, set amount to 0

index = 1234 # identifying index of the asset

# create the asset accept transaction
sp = transaction.SuggestedParams(fee_per_byte, first_valid_round, last_valid_round,␣
↪genesis_hash)
txn = transaction.AssetTransferTxn(address, sp,
                receiver, amount, index)

# sign the transaction
signed_txn = txn.sign(private_key)
```

6.9.7 revoking assets

This transaction must be sent by the asset's clawback manager.

```
from algosdk import account
from algosdk.future import transaction

clawback_private_key = "clawback private key"
clawback_address = "clawback address"

fee_per_byte = 10
first_valid_round = 1000
last_valid_round = 2000
genesis_hash = "SGO1GKSzyE7IEPItTxCByw9x8FmnrCDexi9/cOUJOiI="
receiver = "receiver address" # where to send the revoked assets
target = "revocation target" # address to revoke assets from
amount = 100

index = 1234 # identifying index of the asset

# create the asset transfer transaction
sp = transaction.SuggestedParams(fee_per_byte, first_valid_round, last_valid_round,␣
↪genesis_hash)
txn = transaction.AssetTransferTxn(clawback_address, sp,
                receiver, amount, index, revocation_target=target)

# sign the transaction
signed_txn = txn.sign(clawback_private_key)
```

CHAPTER 7

Rekeying

To rekey an account to a new address, add the `rekey_to` argument to creation.

```
...
# After sending rekeying_txn, every transaction needs to be signed by the SK of the␣
↪following address
rekey_address = "47YPQTIGQEO7T4Y4RWDYWEKV6RTR2UNBQXBABEEGM72ESWDQNCQ52OPASU"
rekeying_txn = transaction.PaymentTxn(sender, sp, receiver, amount, rekey_to=rekey_
↪address)
...
```

CHAPTER 8

Documentation

Documentation for the Python SDK is available at py-algorand-sdk.readthedocs.io.

CHAPTER 9

License

py-algorand-sdk is licensed under a MIT license. See the LICENSE file for details.

CHAPTER 10

Modules

10.1 algosdk

10.1.1 account

generate_account()
Generate an account.

Returns private key, account address

Return type (str, str)

address_from_private_key(*private_key*)
Return the address for the private key.

Parameters **private_key** (*str*) – private key of the account in base64

Returns address of the account

Return type str

10.1.2 algod

class AlgodClient(*algod_token*, *algod_address*, *headers=None*)
Bases: object

Client class for kmd. Handles all algod requests.

Parameters

- **algod_token** (*str*) – algod API token
- **algod_address** (*str*) – algod address
- **headers** (*dict, optional*) – extra header name/value for all requests

algod_token

Type str

algod_address

Type str

headers

Type dict

algod_request(*method*, *requrl*, *params=None*, *data=None*, *headers=None*, *raw_response=False*)
Execute a given request.

Parameters

- **method** (*str*) – request method
- **requrl** (*str*) – url for the request
- **params** (*dict, optional*) – parameters for the request
- **data** (*dict, optional*) – data in the body of the request
- **headers** (*dict, optional*) – additional header for request
- **raw_response** (*bool, default False*) – return the HttpResponse object

Returns loaded from json response body

Return type dict

status(***kwargs*)
Return node status.

health(***kwargs*)
Return null if the node is running.

status_after_block(*block_num*, ***kwargs*)
Return node status immediately after blockNum.

Parameters **block_num** – block number

pending_transactions(*max_txns=0*, ***kwargs*)
Return pending transactions.

Parameters **max_txns** (*int*) – maximum number of transactions to return; if max_txns is 0, return all pending transactions

versions(***kwargs*)
Return algod versions.

ledger_supply(***kwargs*)
Return supply details for node's ledger.

transactions_by_address(*address*, *first=None*, *last=None*, *limit=None*, *from_date=None*, *to_date=None*, ***kwargs*)
Return transactions for an address. If indexer is not enabled, you can search by date and you do not have to specify first and last rounds.

Parameters

- **address** (*str*) – account public key
- **first** (*int, optional*) – no transactions before this block will be returned
- **last** (*int, optional*) – no transactions after this block will be returned; defaults to last round

- **limit** (*int, optional*) – maximum number of transactions to return; default is 100
- **from_date** (*str, optional*) – no transactions before this date will be returned; format YYYY-MM-DD
- **to_date** (*str, optional*) – no transactions after this date will be returned; format YYYY-MM-DD

account_info(*address*, ***kwargs*)
Return account information.

Parameters **address** (*str*) – account public key

asset_info(*index*, ***kwargs*)
Return asset information.

Parameters **index** (*int*) – asset index

list_assets(*max_index=None*, *max_assets=None*, ***kwargs*)
Return a list of up to max_assets assets, where the maximum asset index is max_index.

Parameters

- **max_index** (*int, optional*) – maximum asset index; defaults to 0, which lists most recent assets
- **max_assets** (*int, optional*) – maximum number of assets (0 to 100); defaults to 100

transaction_info(*address*, *transaction_id*, ***kwargs*)
Return transaction information.

Parameters

- **address** (*str*) – account public key
- **transaction_id** (*str*) – transaction ID

pending_transaction_info(*transaction_id*, ***kwargs*)
Return transaction information for a pending transaction.

Parameters **transaction_id** (*str*) – transaction ID

transaction_by_id(*transaction_id*, ***kwargs*)
Return transaction information; only works if indexer is enabled.

Parameters **transaction_id** (*str*) – transaction ID

suggested_fee(***kwargs*)
Return suggested transaction fee.

suggested_params(***kwargs*)
Return suggested transaction parameters.

suggested_params_as_object(***kwargs*)
Return suggested transaction parameters.

send_raw_transaction(*txn*, *headers=None*, ***kwargs*)
Broadcast a signed transaction to the network. Sets the default Content-Type header, if not previously set.

Parameters

- **txn** (*str*) – transaction to send, encoded in base64
- **request_header** (*dict, optional*) – additional header for request

Returns transaction ID

Return type str

send_transaction(*txn*, ***kwargs*)
Broadcast a signed transaction object to the network.

Parameters

- **txn** (*SignedTransaction or MultisigTransaction*) – transaction to send
- **request_header** (*dict, optional*) – additional header for request

Returns transaction ID

Return type str

send_transactions(*txns*, ***kwargs*)
Broadcast list of a signed transaction objects to the network.

Parameters

- **txns** (*SignedTransaction[] or MultisigTransaction[]*) – transactions to send
- **request_header** (*dict, optional*) – additional header for request

Returns first transaction ID

Return type str

block_info(*round*, ***kwargs*)
Return block information.

Parameters **round** (*int*) – block number

block_raw(*round*, ***kwargs*)
Return decoded raw block as the network sees it.

Parameters **round** (*int*) – block number

10.1.3 auction

class Bid(*bidder*, *bid_currency*, *max_price*, *bid_id*, *auction_key*, *auction_id*)
Bases: object

Represents a bid in an auction.

Parameters

- **bidder** (*str*) – address of the bidder
- **bid_currency** (*int*) – how much external currency is being spent
- **max_price** (*int*) – the maximum price the bidder is willing to pay
- **bid_id** (*int*) – bid ID
- **auction_key** (*str*) – address of the auction
- **auction_id** (*int*) – auction ID

bidder

Type str

bid_currency

Type int

max_price

Type int

bid_id

Type int

auction_key

Type str

auction_id

Type int

dictify()

sign(*private_key*)
Sign a bid.

Parameters **private_key** (*str*) – private_key of the bidder

Returns signed bid with the signature

Return type *SignedBid*

static undictify(*d*)

class SignedBid(*bid*, *signature*)
Bases: object

Represents a signed bid in an auction.

Parameters

- **bid** (Bid) – bid that was signed
- **signature** (*str*) – the signature of the bidder

bid

Type *Bid*

signature

Type str

dictify()

static undictify(*d*)

class NoteField(*signed_bid*, *note_field_type*)
Bases: object

Can be encoded and added to a transaction.

Parameters

- **signed_bid** (SignedBid) – bid with signature of bidder
- **note_field_type** (*str*) – the type of note; see constants for possible types

signed_bid

Type *SignedBid*

note_field_type

Type str

dictify()

static undictify(*d*)

10.1.4 constants

Contains useful constants.

kmd_auth_header = 'X-KMD-API-Token'
header key for kmd requests

Type str

algod_auth_header = 'X-Algo-API-Token'
header key for algod requests

Type str

indexer_auth_header = 'X-Indexer-API-Token'
header key for indexer requests

Type str

unversioned_paths = ['/health', '/versions', '/metrics']
paths that don't use the version path prefix

Type str[]

no_auth = []
requests that don't require authentication

Type str[]

payment_txn = 'pay'
indicates a payment transaction

Type str

keyreg_txn = 'keyreg'
indicates a key registration transaction

Type str

assetconfig_txn = 'acfg'
indicates an asset configuration transaction

Type str

assetfreeze_txn = 'afrz'
indicates an asset freeze transaction

Type str

assettransfer_txn = 'axfer'
indicates an asset transfer transaction

Type str

appcall_txn = 'appl'
indicates an app call transaction, allows creating, deleting, and interacting with an application

Type str

note_field_type_deposit = 'd'
indicates a signed deposit in NoteField

Type str

note_field_type_bid = 'b'
indicates a signed bid in NoteField

Type str

note_field_type_settlement = 's'
indicates a signed settlement in NoteField

Type str

note_field_type_params = 'p'
indicates signed params in NoteField

Type str

txid_prefix = b'TX'
transaction prefix when signing

Type bytes

tgid_prefix = b'TG'
transaction group prefix when computing the group ID

Type bytes

bid_prefix = b'aB'
bid prefix when signing

Type bytes

bytes_prefix = b'MX'
bytes prefix when signing

Type bytes

msig_addr_prefix = 'MultisigAddr'
prefix for multisig addresses

Type str

logic_prefix = b'Program'
program (logic) prefix when signing

Type bytes

logic_data_prefix = b'ProgData'
program (logic) data prefix when signing

Type bytes

check_sum_len_bytes = 4
how long checksums should be

Type int

key_len_bytes = 32
how long addresses are in bytes

Type int

address_len = 58
how long addresses are in base32, including the checksum

Type int

mnemonic_len = 25
how long mnemonic phrases are

Type int

min_txn_fee = 1000
minimum transaction fee

Type int

microalgos_to_algos_ratio = 1000000
how many microalgos per algo

Type int

metadata_length = 32
length of asset metadata

Type int

note_max_length = 1024
maximum length of note field

Type int

lease_length = 32
byte length of leases

Type int

multisig_account_limit = 255
maximum number of addresses in a multisig account

Type int

tx_group_limit = 16
maximum number of transaction in a transaction group

Type int

max_asset_decimals = 19
maximum value for decimals in assets

Type int

logic_sig_max_cost = 20000
max execution cost of a teal program

Type int

logic_sig_max_size = 1000
max size of a teal program and its arguments in bytes

Type int

10.1.5 encoding

msgpack_encode(*obj*)
Encode the object using canonical msgpack.

Parameters **obj** (Transaction, SignedTransaction, MultisigTransaction, Multisig, Bid, *or* SignedBid) – object to be encoded

Returns msgpack encoded object

Return type str

Note: Canonical Msgpack: maps must contain keys in lexicographic order; maps must omit key-value pairs where the value is a zero-value; positive integer values must be encoded as "unsigned" in msgpack, regardless of whether the value space is semantically signed or unsigned; integer values must be represented in the shortest possible encoding; binary arrays must be represented using the "bin" format family (that is, use the most recent version of msgpack rather than the older msgpack version that had no "bin" family).

---

future_msgpack_decode(*enc*)
Decode a msgpack encoded object from a string.

Parameters **enc** (*str*) – string to be decoded

Returns decoded object

Return type *Transaction*, *SignedTransaction*, *Multisig*, *Bid*, or *SignedBid*

msgpack_decode(*enc*)
Decode a msgpack encoded object from a string.

Parameters **enc** (*str*) – string to be decoded

Returns decoded object

Return type *Transaction*, *SignedTransaction*, *Multisig*, *Bid*, or *SignedBid*

is_valid_address(*addr*)
Check if the string address is a valid Algorand address.

Parameters **addr** (*str*) – base32 address

Returns whether or not the address is valid

Return type bool

decode_address(*addr*)
Decode a string address into its address bytes and checksum.

Parameters **addr** (*str*) – base32 address

Returns address decoded into bytes

Return type bytes

encode_address(*addr_bytes*)
Encode a byte address into a string composed of the encoded bytes and the checksum.

Parameters **addr_bytes** (*bytes*) – address in bytes

Returns base32 encoded address

Return type str

checksum(*data*)
Compute the checksum of arbitrary binary input.

Parameters **data** (*bytes*) – data as bytes

Returns checksum of the data

Return type bytes

10.1.6 error

exception BadTxnSenderError
Bases: Exception

exception InvalidThresholdError
Bases: Exception

exception InvalidSecretKeyError
Bases: Exception

exception MergeKeysMismatchError
Bases: Exception

exception DuplicateSigMismatchError
Bases: Exception

exception WrongAmountType
Bases: Exception

exception WrongChecksumError
Bases: Exception

exception WrongKeyLengthError
Bases: Exception

exception WrongMnemonicLengthError
Bases: Exception

exception WrongKeyBytesLengthError
Bases: Exception

exception UnknownMsigVersionError
Bases: Exception

exception WrongMetadataLengthError
Bases: Exception

exception WrongLeaseLengthError
Bases: Exception

exception WrongNoteType
Bases: Exception

exception WrongNoteLength
Bases: Exception

exception InvalidProgram(*message='invalid program for logic sig'*)
Bases: Exception

exception TransactionGroupSizeError
Bases: Exception

exception MultisigAccountSizeError
Bases: Exception

exception OutOfRangeDecimalsError
Bases: Exception

exception EmptyAddressError
Bases: Exception

exception WrongContractError(*contract_type*)
Bases: Exception

exception TemplateInputError
Bases: Exception

exception TemplateError
Bases: Exception

exception KMDHTTPError
Bases: Exception

exception AlgodHTTPError(*msg*, *code=None*)
Bases: Exception

exception IndexerHTTPError
Bases: Exception

10.1.7 future

10.1.7.1 future.template

class Template
Bases: object

get_address()
Return the address of the contract.

get_program()

class Split(*owner: str*, *receiver_1: str*, *receiver_2: str*, *rat_1: int*, *rat_2: int*, *expiry_round: int*, *min_pay: int*, *max_fee: int*)
Bases: *algosdk.future.template.Template*

Split allows locking algos in an account which allows transfering to two predefined addresses in a specified ratio such that for the given ratn and ratd parameters we have:

first_recipient_amount * rat_2 == second_recipient_amount * rat_1

Split also has an expiry round, after which the owner can transfer back the funds.

Parameters

- **owner** (*str*) – an address that can receive the funds after the expiry round
- **receiver_1** (*str*) – first address to receive funds
- **receiver_2** (*str*) – second address to receive funds
- **rat_1** (*int*) – how much receiver_1 receives (proportionally)
- **rat_2** (*int*) – how much receiver_2 receives (proportionally)
- **expiry_round** (*int*) – the round on which the funds can be transferred back to owner
- **min_pay** (*int*) – the minimum number of microalgos that can be transferred from the account to receiver_1
- **max_fee** (*int*) – half the maximum fee that can be paid to the network by the account

get_program()
Return a byte array to be used in LogicSig.

static get_split_funds_transaction(*contract*, *amount: int*, *sp*)
Return a group transactions array which transfers funds according to the contract's ratio.

Parameters

- **amount** (*int*) – total amount to be transferred
- **sp** (SuggestedParams) – suggested params from algod

Returns Transaction[]

class HTLC(*owner: str*, *receiver: str*, *hash_function: str*, *hash_image: str*, *expiry_round: int*, *max_fee: int*)

Bases: *algosdk.future.template.Template*

Hash Time Locked Contract allows a user to recieve the Algo prior to a deadline (in terms of a round) by proving knowledge of a special value or to forfeit the ability to claim, returning it to the payer. This contract is usually used to perform cross-chained atomic swaps.

More formally, algos can be transfered under only two circumstances:

1. To receiver if hash_function(arg_0) = hash_value
2. To owner if txn.FirstValid > expiry_round

Parameters

- **owner** (*str*) – an address that can receive the asset after the expiry round
- **receiver** (*str*) – address to receive Algos
- **hash_function** (*str*) – the hash function to be used (must be either sha256 or keccak256)
- **hash_image** (*str*) – the hash image in base64
- **expiry_round** (*int*) – the round on which the assets can be transferred back to owner
- **max_fee** (*int*) – the maximum fee that can be paid to the network by the account

get_program()

Return a byte array to be used in LogicSig.

static get_transaction(*contract*, *preimage*, *sp*)

Return a transaction which will release funds if a matching preimage is used.

Parameters

- **contract** (*bytes*) – the contract containing information, should be received from payer
- **preimage** (*str*) – the preimage of the hash in base64
- **sp** (SuggestedParams) – suggested params from algod

Returns

transaction to claim algos from contract account

Return type *LogicSigTransaction*

class DynamicFee(*receiver: str*, *amount: int*, *sp*, *close_remainder_address: str = None*)

Bases: *algosdk.future.template.Template*

DynamicFee contract allows you to create a transaction without specifying the fee. The fee will be determined at the moment of transfer.

Parameters

- **receiver** (*str*) – address to receive the assets
- **amount** (*int*) – amount of assets to transfer
- **sp** (SuggestedParams) – suggested params from algod

- **close_remainder_address** (*str, optional*) – the address that recieves the remainder

get_program()
Return a byte array to be used in LogicSig.

static get_transactions(*txn*, *lsig*, *private_key*, *fee*)
Create and sign the secondary dynamic fee transaction, update transaction fields, and sign as the fee payer; return both transactions.

Parameters

- **txn** (Transaction) – main transaction from payer
- **lsig** (LogicSig) – signed logic received from payer
- **private_key** (*str*) – the secret key of the account that pays the fee in base64
- **fee** (*int*) – fee per byte, for both transactions

sign_dynamic_fee(*private_key*)
Return the main transaction and signed logic needed to complete the transfer. These should be sent to the fee payer, who can use get_transactions() to update fields and create the auxiliary transaction.

Parameters **private_key** (*bytes*) – the secret key to sign the contract in base64

class PeriodicPayment(*receiver: str*, *amount: int*, *withdrawing_window: int*, *period: int*, *max_fee: int*, *timeout: int*)
Bases: *algosdk.future.template.Template*

PeriodicPayment contract enables creating an account which allows the withdrawal of a fixed amount of assets every fixed number of rounds to a specific Algrorand Address. In addition, the contract allows to add timeout, after which the address can withdraw the rest of the assets.

Parameters

- **receiver** (*str*) – address to receive the assets
- **amount** (*int*) – amount of assets to transfer at every cycle
- **withdrawing_window** (*int*) – the number of blocks in which the user can withdraw the asset once the period start (must be < 1000)
- **period** (*int*) – how often the address can withdraw assets (in rounds)
- **fee** (*int*) – maximum fee per transaction
- **timeout** (*int*) – a round in which the receiver can withdraw the rest of the funds after

get_program()
Return a byte array to be used in LogicSig.

static get_withdrawal_transaction(*contract*, *sp*)
Return the withdrawal transaction to be sent to the network.

Parameters

- **contract** (*bytes*) – contract containing information, should be received from payer
- **sp** (SuggestedParams) – suggested params from algod; the value of sp.last will not be used. Instead, the last valid round will be calculated from first valid round and withdrawing window

class LimitOrder(*owner: str*, *asset_id: int*, *ratn: int*, *ratd: int*, *expiry_round: int*, *max_fee: int*, *min_trade: int*)
Bases: *algosdk.future.template.Template*

Limit Order allows to trade Algos for other assets given a specific ratio; for N Algos, swap for Rate * N Assets.
...

Parameters

- **owner** (*str*) – an address that can receive the asset after the expiry round
- **asset_id** (*int*) – asset to be transfered
- **ratn** (*int*) – the numerator of the exchange rate
- **ratd** (*int*) – the denominator of the exchange rate
- **expiry_round** (*int*) – the round on which the assets can be transferred back to owner
- **max_fee** (*int*) – the maximum fee that can be paid to the network by the account
- **min_trade** (*int*) – the minimum amount (of Algos) to be traded away

get_program()
Return a byte array to be used in LogicSig.

static get_swap_assets_transactions(*contract: bytes*, *asset_amount: int*, *microalgo_amount: int*, *private_key: str*, *sp*)
Return a group transactions array which transfer funds according to the contract's ratio.

Parameters

- **contract** (*bytes*) – the contract containing information, should be received from payer
- **asset_amount** (*int*) – the amount of assets to be sent
- **microalgo_amount** (*int*) – the amount of microalgos to be received
- **private_key** (*str*) – the secret key to sign the contract
- **sp** (SuggestedParams) – suggested params from algod

put_uvarint(*buf*, *x*)

inject(*orig*, *offsets*, *values*, *values_types*)

10.1.7.2 future.transaction

class SuggestedParams(*fee*, *first*, *last*, *gh*, *gen=None*, *flat_fee=False*, *consensus_version=None*, *min_fee=None*)
Bases: object

Contains various fields common to all transaction types.

Parameters

- **fee** (*int*) – transaction fee (per byte)
- **first** (*int*) – first round for which the transaction is valid
- **last** (*int*) – last round for which the transaction is valid
- **gh** (*str*) – genesis hash
- **gen** (*str, optional*) – genesis id
- **flat_fee** (*bool, optional*) – whether the specified fee is a flat fee
- **consensus_version** (*str, optional*) – the consensus protocol version as of 'first'
- **min_fee** (*int, optional*) – the minimum transaction fee (flat)

fee

Type int

first

Type int

last

Type int

gen

Type str

gh

Type str

flat_fee

Type bool

consensus_version

Type str

min_fee

Type int

class Transaction(*sender*, *sp*, *note*, *lease*, *txn_type*, *rekey_to*)

Bases: object

Superclass for various transaction types.

get_txid()

Get the transaction's ID.

Returns transaction ID

Return type str

sign(*private_key*)

Sign the transaction with a private key.

Parameters **private_key** (*str*) – the private key of the signing account

Returns signed transaction with the signature

Return type *SignedTransaction*

raw_sign(*private_key*)

Sign the transaction.

Parameters **private_key** (*str*) – the private key of the signing account

Returns signature

Return type bytes

estimate_size()

dictify()

static undictify(*d*)

class PaymentTxn(*sender*, *sp*, *receiver*, *amt*, *close_remainder_to=None*, *note=None*, *lease=None*, *rekey_to=None*)

Bases: *algosdk.future.transaction.Transaction*

Represents a payment transaction.

Parameters

- **sender** (*str*) – address of the sender
- **sp** (SuggestedParams) – suggested params from algod
- **receiver** (*str*) – address of the receiver
- **amt** (*int*) – amount in microAlgos to be sent
- **close_remainder_to** (*str, optional*) – if nonempty, account will be closed and remaining algos will be sent to this address
- **note** (*bytes, optional*) – arbitrary optional bytes
- **lease** (*byte[32], optional*) – specifies a lease, and no other transaction with the same sender and lease can be confirmed in this transaction's valid rounds
- **rekey_to** (*str, optional*) – additionally rekey the sender to this address

sender

Type str

fee

Type int

first_valid_round

Type int

last_valid_round

Type int

note

Type bytes

genesis_id

Type str

genesis_hash

Type str

group

Type bytes

receiver

Type str

amt

Type int

close_remainder_to

Type str

type

Type str

lease

Type byte[32]

rekey_to

Type str

dictify()

class KeyregTxn(*sender*, *sp*, *votekey*, *selkey*, *votefst*, *votelst*, *votekd*, *note=None*, *lease=None*, *rekey_to=None*)

Bases: *algosdk.future.transaction.Transaction*

Represents a key registration transaction.

Parameters

- **sender** (*str*) – address of sender
- **sp** (SuggestedParams) – suggested params from algod
- **votekey** (*str*) – participation public key in base64
- **selkey** (*str*) – VRF public key in base64
- **votefst** (*int*) – first round to vote
- **votelst** (*int*) – last round to vote
- **votekd** (*int*) – vote key dilution
- **note** (*bytes, optional*) – arbitrary optional bytes
- **lease** (*byte[32], optional*) – specifies a lease, and no other transaction with the same sender and lease can be confirmed in this transaction's valid rounds
- **rekey_to** (*str, optional*) – additionally rekey the sender to this address

sender

Type str

fee

Type int

first_valid_round

Type int

last_valid_round

Type int

note

Type bytes

genesis_id

Type str

genesis_hash

Type str

group

Type bytes

votepk

Type str

selkey

Type str

votefst

Type int

votelst

Type int

votekd

Type int

type

Type str

lease

Type byte[32]

rekey_to

Type str

dictify()

class AssetConfigTxn(*sender*, *sp*, *index=None*, *total=None*, *default_frozen=None*, *unit_name=None*, *asset_name=None*, *manager=None*, *reserve=None*, *freeze=None*, *clawback=None*, *url=None*, *metadata_hash=None*, *note=None*, *lease=None*, *strict_empty_address_check=True*, *decimals=0*, *rekey_to=None*)

Bases: *algosdk.future.transaction.Transaction*

Represents a transaction for asset creation, reconfiguration, or destruction.

To create an asset, include the following: total, default_frozen, unit_name, asset_name, manager, reserve, freeze, clawback, url, metadata, decimals

To destroy an asset, include the following: index, strict_empty_address_check (set to False)

To update asset configuration, include the following: index, manager, reserve, freeze, clawback, strict_empty_address_check (optional)

Parameters

- **sender** (*str*) – address of the sender
- **sp** (SuggestedParams) – suggested params from algod
- **index** (*int, optional*) – index of the asset
- **total** (*int, optional*) – total number of base units of this asset created
- **default_frozen** (*bool, optional*) – whether slots for this asset in user accounts are frozen by default
- **unit_name** (*str, optional*) – hint for the name of a unit of this asset
- **asset_name** (*str, optional*) – hint for the name of the asset

- **manager** (*str, optional*) – address allowed to change nonzero addresses for this asset
- **reserve** (*str, optional*) – account whose holdings of this asset should be reported as "not minted"
- **freeze** (*str, optional*) – account allowed to change frozen state of holdings of this asset
- **clawback** (*str, optional*) – account allowed take units of this asset from any account
- **url** (*str, optional*) – a URL where more information about the asset can be retrieved
- **metadata_hash** (*byte[32], optional*) – a commitment to some unspecified asset metadata (32 byte hash)
- **note** (*bytes, optional*) – arbitrary optional bytes
- **lease** (*byte[32], optional*) – specifies a lease, and no other transaction with the same sender and lease can be confirmed in this transaction's valid rounds
- **strict_empty_address_check** (*bool, optional*) – set this to False if you want to specify empty addresses. Otherwise, if this is left as True (the default), having empty addresses will raise an error, which will prevent accidentally removing admin access to assets or deleting the asset.
- **decimals** (*int, optional*) – number of digits to use for display after decimal. If set to 0, the asset is not divisible. If set to 1, the base unit of the asset is in tenths. Must be between 0 and 19, inclusive. Defaults to 0.
- **rekey_to** (*str, optional*) – additionally rekey the sender to this address

sender

Type str

fee

Type int

first_valid_round

Type int

last_valid_round

Type int

genesis_hash

Type str

index

Type int

total

Type int

default_frozen

Type bool

unit_name

Type str

asset_name

Type str

manager

Type str

reserve

Type str

freeze

Type str

clawback

Type str

url

Type str

metadata_hash

Type byte[32]

note

Type bytes

genesis_id

Type str

type

Type str

lease

Type byte[32]

decimals

Type int

rekey

Type str

dictify()

class AssetFreezeTxn(*sender*, *sp*, *index*, *target*, *new_freeze_state*, *note=None*, *lease=None*, *rekey_to=None*)

Bases: *algosdk.future.transaction.Transaction*

Represents a transaction for freezing or unfreezing an account's asset holdings. Must be issued by the asset's freeze manager.

Parameters

- **sender** (*str*) – address of the sender, who must be the asset's freeze manager
- **sp** (SuggestedParams) – suggested params from algod
- **index** (*int*) – index of the asset

- **target** (*str*) – address having its assets frozen or unfrozen
- **new_freeze_state** (*bool*) – true if the assets should be frozen, false if they should be transferrable
- **note** (*bytes, optional*) – arbitrary optional bytes
- **lease** (*byte[32], optional*) – specifies a lease, and no other transaction with the same sender and lease can be confirmed in this transaction's valid rounds
- **rekey_to** (*str, optional*) – additionally rekey the sender to this address

sender

Type str

fee

Type int

first_valid_round

Type int

last_valid_round

Type int

genesis_hash

Type str

index

Type int

target

Type str

new_freeze_state

Type bool

note

Type bytes

genesis_id

Type str

type

Type str

lease

Type byte[32]

rekey_to

Type str

dictify()

class AssetTransferTxn(*sender*, *sp*, *receiver*, *amt*, *index*, *close_assets_to=None*, *revocation_target=None*, *note=None*, *lease=None*, *rekey_to=None*)

Bases: *algosdk.future.transaction.Transaction*

Represents a transaction for asset transfer.

To begin accepting an asset, supply the same address as both sender and receiver, and set amount to 0.

To revoke an asset, set revocation_target, and issue the transaction from the asset's revocation manager account.

Parameters

- **sender** (*str*) – address of the sender
- **sp** (SuggestedParams) – suggested params from algod
- **receiver** (*str*) – address of the receiver
- **amt** (*int*) – amount of asset base units to send
- **index** (*int*) – index of the asset
- **close_assets_to** (*string, optional*) – send all of sender's remaining assets, after paying *amt* to receiver, to this address
- **revocation_target** (*string, optional*) – send assets from this address, rather than the sender's address (can only be used by an asset's revocation manager, also known as clawback)
- **note** (*bytes, optional*) – arbitrary optional bytes
- **lease** (*byte[32], optional*) – specifies a lease, and no other transaction with the same sender and lease can be confirmed in this transaction's valid rounds
- **rekey_to** (*str, optional*) – additionally rekey the sender to this address

sender

Type str

fee

Type int

first_valid_round

Type int

last_valid_round

Type int

genesis_hash

Type str

index

Type int

amount

Type int

receiver

Type string

close_assets_to

Type string

revocation_target

Type string

note

Type bytes

genesis_id

Type str

type

Type str

lease

Type byte[32]

rekey_to

Type str

dictify()

class StateSchema(*num_uints=None*, *num_byte_slices=None*)

Bases: object

Restricts state for an application call.

Parameters

- **num_uints** (*int, optional*) – number of uints to store
- **num_byte_slices** (*int, optional*) – number of byte slices to store

num_uints

Type int

num_byte_slices

Type int

dictify()

static undictify(*d*)

class OnComplete

Bases: enum.IntEnum

An enumeration.

NoOpOC = 0

OptInOC = 1

CloseOutOC = 2

ClearStateOC = 3

UpdateApplicationOC = 4

DeleteApplicationOC = 5

class ApplicationCallTxn(*sender*, *sp*, *index*, *on_complete*, *local_schema=None*, *global_schema=None*, *approval_program=None*, *clear_program=None*, *app_args=None*, *accounts=None*, *foreign_apps=None*, *foreign_assets=None*, *note=None*, *lease=None*, *rekey_to=None*)

Bases: *algosdk.future.transaction.Transaction*

Represents a transaction that interacts with the application system.

Parameters

- **sender** (*str*) – address of the sender
- **sp** (SuggestedParams) – suggested params from algod
- **index** (*int*) – index of the application to call; 0 if creating a new application
- **on_complete** (OnComplete) – intEnum representing what app should do on completion
- **local_schema** (StateSchema, *optional*) – restricts what can be stored by created application; must be omitted if not creating an application
- **global_schema** (StateSchema, *optional*) – restricts what can be stored by created application; must be omitted if not creating an application
- **approval_program** (*bytes, optional*) – the program to run on transaction approval; must be omitted if not creating or updating an application
- **clear_program** (*bytes, optional*) – the program to run when state is being cleared; must be omitted if not creating or updating an application
- **app_args** (*list[bytes], optional*) – list of arguments to the application, each argument itself a buf
- **accounts** (*list[string], optional*) – list of additional accounts involved in call
- **foreign_apps** (*list[int], optional*) – list of other applications (identified by index) involved in call
- **foreign_assets** (*list[int], optional*) – list of assets involved in call

sender

Type str

fee

Type int

first_valid_round

Type int

last_valid_round

Type int

genesis_hash

Type str

index

Type int

on_complete

Type int

local_schema

Type *StateSchema*

global_schema

Type *StateSchema*

approval_program

Type bytes

clear_program

Type bytes

app_args

Type list[bytes]

accounts

Type list[str]

foreign_apps

Type list[int]

foreign_assets

Type list[int]

dictify()

class ApplicationCreateTxn(*sender*, *sp*, *on_complete*, *approval_program*, *clear_program*, *global_schema*, *local_schema*, *app_args=None*, *accounts=None*, *foreign_apps=None*, *foreign_assets=None*, *note=None*, *lease=None*, *rekey_to=None*)

Bases: *algosdk.future.transaction.ApplicationCallTxn*

Make a transaction that will create an application.

Parameters

- **sender** (*str*) – address of sender
- **sp** (SuggestedParams) – contains information such as fee and genesis hash
- **on_complete** (OnComplete) – what application should so once the program is done being run
- **approval_program** (*bytes*) – the compiled TEAL that approves a transaction
- **clear_program** (*bytes*) – the compiled TEAL that runs when clearing state
- **global_schema** (StateSchema) – restricts the number of ints and byte slices in the global state
- **local_schema** (StateSchema) – restructs the number of ints and byte slices in the per-user local state
- **app_args** (*list[bytes], optional*) – any additional arguments to the application
- **accounts** (*list[str], optional*) – any additional accounts to supply to the application
- **foreign_apps** (*list[int], optional*) – any other apps used by the application, identified by app index
- **foreign_assets** (*list[int], optional*) – list of assets involved in call
- **note** (*bytes, optional*) – transaction note field
- **lease** (*bytes, optional*) – transaction lease field
- **rekey_to** (*str, optional*) – rekey-to field, see Transaction

See ApplicationCallTxn

dictify()

class ApplicationUpdateTxn(*sender*, *sp*, *index*, *approval_program*, *clear_program*, *app_args=None*, *accounts=None*, *foreign_apps=None*, *foreign_assets=None*, *note=None*, *lease=None*, *rekey_to=None*)

Bases: *algosdk.future.transaction.ApplicationCallTxn*

Make a transaction that will change an application's approval and clear programs.

Parameters

- **sender** (*str*) – address of sender
- **sp** (SuggestedParams) – contains information such as fee and genesis hash
- **index** (*int*) – the application to update
- **approval_program** (*bytes*) – the new compiled TEAL that approves a transaction
- **clear_program** (*bytes*) – the new compiled TEAL that runs when clearing state
- **app_args** (*list[bytes], optional*) – any additional arguments to the application
- **accounts** (*list[str], optional*) – any additional accounts to supply to the application
- **foreign_apps** (*list[int], optional*) – any other apps used by the application, identified by app index
- **foreign_assets** (*list[int], optional*) – list of assets involved in call
- **note** (*bytes, optional*) – transaction note field
- **lease** (*bytes, optional*) – transaction lease field
- **rekey_to** (*str, optional*) – rekey-to field, see Transaction

See ApplicationCallTxn

dictify()

class ApplicationDeleteTxn(*sender*, *sp*, *index*, *app_args=None*, *accounts=None*, *foreign_apps=None*, *foreign_assets=None*, *note=None*, *lease=None*, *rekey_to=None*)

Bases: *algosdk.future.transaction.ApplicationCallTxn*

Make a transaction that will delete an application

Parameters

- **sender** (*str*) – address of sender
- **sp** (SuggestedParams) – contains information such as fee and genesis hash
- **index** (*int*) – the application to update
- **app_args** (*list[bytes], optional*) – any additional arguments to the application
- **accounts** (*list[str], optional*) – any additional accounts to supply to the application
- **foreign_apps** (*list[int], optional*) – any other apps used by the application, identified by app index
- **foreign_assets** (*list[int], optional*) – list of assets involved in call
- **note** (*bytes, optional*) – transaction note field

- **lease** (*bytes, optional*) – transaction lease field
- **rekey_to** (*str, optional*) – rekey-to field, see Transaction

See ApplicationCallTxn

dictify()

class ApplicationOptInTxn(*sender*, *sp*, *index*, *app_args=None*, *accounts=None*, *foreign_apps=None*, *foreign_assets=None*, *note=None*, *lease=None*, *rekey_to=None*)

Bases: *algosdk.future.transaction.ApplicationCallTxn*

Make a transaction that will opt in to an application

Parameters

- **sender** (*str*) – address of sender
- **sp** (SuggestedParams) – contains information such as fee and genesis hash
- **index** (*int*) – the application to update
- **app_args** (*list[bytes], optional*) – any additional arguments to the application
- **accounts** (*list[str], optional*) – any additional accounts to supply to the application
- **foreign_apps** (*list[int], optional*) – any other apps used by the application, identified by app index
- **foreign_assets** (*list[int], optional*) – list of assets involved in call
- **note** (*bytes, optional*) – transaction note field
- **lease** (*bytes, optional*) – transaction lease field
- **rekey_to** (*str, optional*) – rekey-to field, see Transaction

See ApplicationCallTxn

dictify()

class ApplicationCloseOutTxn(*sender*, *sp*, *index*, *app_args=None*, *accounts=None*, *foreign_apps=None*, *foreign_assets=None*, *note=None*, *lease=None*, *rekey_to=None*)

Bases: *algosdk.future.transaction.ApplicationCallTxn*

Make a transaction that will close out a user's state in an application

Parameters

- **sender** (*str*) – address of sender
- **sp** (SuggestedParams) – contains information such as fee and genesis hash
- **index** (*int*) – the application to update
- **app_args** (*list[bytes], optional*) – any additional arguments to the application
- **accounts** (*list[str], optional*) – any additional accounts to supply to the application
- **foreign_apps** (*list[int], optional*) – any other apps used by the application, identified by app index
- **foreign_assets** (*list[int], optional*) – list of assets involved in call
- **note** (*bytes, optional*) – transaction note field

- **lease** (*bytes, optional*) – transaction lease field
- **rekey_to** (*str, optional*) – rekey-to field, see Transaction

See ApplicationCallTxn

dictify()

class ApplicationClearStateTxn(*sender*, *sp*, *index*, *app_args=None*, *accounts=None*, *foreign_apps=None*, *foreign_assets=None*, *note=None*, *lease=None*, *rekey_to=None*)

Bases: *algosdk.future.transaction.ApplicationCallTxn*

Make a transaction that will clear a user's state an application

Parameters

- **sender** (*str*) – address of sender
- **sp** (SuggestedParams) – contains information such as fee and genesis hash
- **index** (*int*) – the application to update
- **app_args** (*list[bytes], optional*) – any additional arguments to the application
- **accounts** (*list[str], optional*) – any additional accounts to supply to the application
- **foreign_apps** (*list[int], optional*) – any other apps used by the application, identified by app index
- **foreign_assets** (*list[int], optional*) – list of assets involved in call
- **note** (*bytes, optional*) – transaction note field
- **lease** (*bytes, optional*) – transaction lease field
- **rekey_to** (*str, optional*) – rekey-to field, see Transaction

See ApplicationCallTxn

dictify()

class ApplicationNoOpTxn(*sender*, *sp*, *index*, *app_args=None*, *accounts=None*, *foreign_apps=None*, *foreign_assets=None*, *note=None*, *lease=None*, *rekey_to=None*)

Bases: *algosdk.future.transaction.ApplicationCallTxn*

Make a transaction that will do nothing on application completion In other words, just call the application

Parameters

- **sender** (*str*) – address of sender
- **sp** (SuggestedParams) – contains information such as fee and genesis hash
- **index** (*int*) – the application to update
- **app_args** (*list[bytes], optional*) – any additional arguments to the application
- **accounts** (*list[str], optional*) – any additional accounts to supply to the application
- **foreign_apps** (*list[int], optional*) – any other apps used by the application, identified by app index
- **foreign_assets** (*list[int], optional*) – list of assets involved in call
- **note** (*bytes, optional*) – transaction note field

- **lease** (*bytes, optional*) – transaction lease field
- **rekey_to** (*str, optional*) – rekey-to field, see Transaction

See ApplicationCallTxn

dictify()

class SignedTransaction(*transaction*, *signature*, *authorizing_address=None*)
Bases: object

Represents a signed transaction.

Parameters

- **transaction** (Transaction) – transaction that was signed
- **signature** (*str*) – signature of a single address
- **authorizing_address** (*str, optional*) – the address authorizing the signed transaction, if different from sender

transaction

Type *Transaction*

signature

Type str

authorizing_address

Type str

get_txid()
Get the transaction's ID.

Returns transaction ID

Return type str

dictify()

static undictify(*d*)

class MultisigTransaction(*transaction*, *multisig*)
Bases: object

Represents a signed transaction.

Parameters

- **transaction** (Transaction) – transaction that was signed
- **multisig** (Multisig) – multisig account and signatures

transaction

Type *Transaction*

multisig

Type *Multisig*

sign(*private_key*)
Sign the multisig transaction.

Parameters **private_key** (*str*) – private key of signing account

Note: A new signature will replace the old if there is already a signature for the address. To sign another transaction, you can either overwrite the signatures in the current Multisig, or you can use Multisig.get_multisig_account() to get a new multisig object with the same addresses.

get_txid()
Get the transaction's ID.

Returns transaction ID

Return type str

dictify()

static undictify(*d*)

static merge(*part_stxs*)
Merge partially signed multisig transactions.

Parameters **part_stxs** (*MultisigTransaction[]*) – list of partially signed multisig transactions

Returns multisig transaction containing signatures

Return type *MultisigTransaction*

Note: Only use this if you are given two partially signed multisig transactions. To append a signature to a multisig transaction, just use MultisigTransaction.sign()

class Multisig(*version*, *threshold*, *addresses*)
Bases: object

Represents a multisig account and signatures.

Parameters

- **version** (*int*) – currently, the version is 1
- **threshold** (*int*) – how many signatures are necessary
- **addresses** (*str[]*) – addresses in the multisig account

version

Type int

threshold

Type int

subsigs

Type *MultisigSubsig*[]

validate()
Check if the multisig account is valid.

address()
Return the multisig account address.

verify(*message*)
Verify that the multisig is valid for the message.

dictify()

json_dictify()

static undictify(*d*)

get_multisig_account()
Return a Multisig object without signatures.

get_public_keys()
Return the base32 encoded addresses for the multisig account.

class MultisigSubsig(*public_key*, *signature=None*)
Bases: object

public_key

Type bytes

signature

Type bytes

dictify()

json_dictify()

static undictify(*d*)

class LogicSig(*program*, *args=None*)
Bases: object

Represents a logic signature

Parameters

- **logic** (*bytes*) – compiled program
- **args** (*list[bytes]*) – args are not signed, but are checked by logic

logic

Type bytes

sig

Type bytes

msig

Type *Multisig*

args

Type list[bytes]

dictify()

static undictify(*d*)

verify(*public_key*)
Verifies LogicSig against the transaction's sender address

Parameters **public_key** (*bytes*) – sender address

Returns true if the signature is valid (the sender address matches the logic hash or the signature is valid against the sender address), false otherwise

Return type bool

address()
Compute hash of the logic sig program (that is the same as escrow account address) as string address

Returns program address

Return type str

static sign_program(*program*, *private_key*)

static single_sig_multisig(*program*, *private_key*, *multisig*)

sign(*private_key*, *multisig=None*)
Creates signature (if no pk provided) or multi signature

Parameters

- **private_key** (*str*) – private key of signing account
- **multisig** (Multisig) – optional multisig account without signatures to sign with

Raises InvalidSecretKeyError – if no matching private key in multisig object

append_to_multisig(*private_key*)
Appends a signature to multi signature

Parameters **private_key** (*str*) – private key of signing account

Raises InvalidSecretKeyError – if no matching private key in multisig object

class LogicSigTransaction(*transaction*, *lsig*)
Bases: object

Represents a logic signed transaction

Parameters

- **transaction** (Transaction) –
- **lsig** (LogicSig) –

transaction

Type *Transaction*

lsig

Type *LogicSig*

verify()
Verify LogicSig against the transaction

Returns true if the signature is valid (the sender address matches the logic hash or the signature is valid against the sender address), false otherwise

Return type bool

get_txid()
Get the transaction's ID.

Returns transaction ID

Return type str

dictify()

static undictify(*d*)

write_to_file(*txns*, *path*, *overwrite=True*)
Write signed or unsigned transactions to a file.

Parameters

- **txns** (*Transaction[], SignedTransaction[], or MultisigTransaction[]*) – can be a mix of the three
- **path** (*str*) – file to write to
- **overwrite** (*bool*) – whether or not to overwrite what's already in the file; if False, transactions will be appended to the file

Returns true if the transactions have been written to the file

Return type bool

retrieve_from_file(*path*)
Retrieve signed or unsigned transactions from a file.

Parameters **path** (*str*) – file to read from

Returns can be a mix of the three

Return type *Transaction*[], *SignedTransaction*[], or *MultisigTransaction*[]

class TxGroup(*txns*)
Bases: object

dictify()

static undictify(*d*)

calculate_group_id(*txns*)
Calculate group id for a given list of unsigned transactions

Parameters **txns** (*list*) – list of unsigned transactions

Returns checksum value representing the group id

Return type bytes

assign_group_id(*txns*, *address=None*)
Assign group id to a given list of unsigned transactions

Parameters

- **txns** (*list*) – list of unsigned transactions
- **address** (*str*) – optional sender address specifying which transaction return

Returns list of unsigned transactions with group property set

Return type txns (list)

10.1.8 kmd

class KMDClient(*kmd_token*, *kmd_address*)
Bases: object

Client class for kmd. Handles all kmd requests.

Parameters

- **kmd_token** (*str*) – kmd API token
- **kmd_address** (*str*) – kmd address

kmd_token

Type str

kmd_address

Type str

kmd_request(*method*, *requrl*, *params=None*, *data=None*)
Execute a given request.

Parameters

- **method** (*str*) – request method
- **requrl** (*str*) – url for the request
- **params** (*dict, optional*) – parameters for the request
- **data** (*dict, optional*) – data in the body of the request

Returns loaded from json response body

Return type dict

versions()
Get kmd versions.

Returns list of versions

Return type str[]

list_wallets()
List all wallets hosted on node.

Returns list of dictionaries containing wallet information

Return type dict[]

create_wallet(*name*, *pswd*, *driver_name='sqlite'*, *master_deriv_key=None*)
Create a new wallet.

Parameters

- **name** (*str*) – wallet name
- **pswd** (*str*) – wallet password
- **driver_name** (*str, optional*) – name of the driver
- **master_deriv_key** (*str, optional*) – if recovering a wallet, include

Returns dictionary containing wallet information

Return type dict

get_wallet(*handle*)
Get wallet information.

Parameters **handle** (*str*) – wallet handle token

Returns dictionary containing wallet handle and wallet information

Return type dict

init_wallet_handle(*id*, *password*)
Initialize a handle for the wallet.

Parameters

- **id** (*str*) – wallet ID

- **password** (*str*) – wallet password

Returns wallet handle token

Return type str

release_wallet_handle(*handle*)
Deactivate the handle for the wallet.

Args: handle (str): wallet handle token

Returns True if the handle has been deactivated

Return type bool

renew_wallet_handle(*handle*)
Renew the wallet handle.

Parameters **handle** (*str*) – wallet handle token

Returns dictionary containing wallet handle and wallet information

Return type dict

rename_wallet(*id*, *password*, *new_name*)
Rename the wallet.

Parameters

- **id** (*str*) – wallet ID
- **password** (*str*) – wallet password
- **new_name** (*str*) – new name for the wallet

Returns dictionary containing wallet information

Return type dict

export_master_derivation_key(*handle*, *password*)
Get the wallet's master derivation key.

Parameters

- **handle** (*str*) – wallet handle token
- **password** (*str*) – wallet password

Returns master derivation key

Return type str

import_key(*handle*, *private_key*)
Import an account into the wallet.

Parameters

- **handle** (*str*) – wallet handle token
- **private_key** (*str*) – private key of account to be imported

Returns base32 address of the account

Return type str

export_key(*handle*, *password*, *address*)
Return an account private key.

Parameters

- **handle** (*str*) – wallet handle token
- **password** (*str*) – wallet password
- **address** (*str*) – base32 address of the account

Returns private key

Return type str

generate_key(*handle*, *display_mnemonic=True*)
Generate a key in the wallet.

Parameters

- **handle** (*str*) – wallet handle token
- **display_mnemonic** (*bool, optional*) – whether or not the mnemonic should be displayed

Returns base32 address of the generated account

Return type str

delete_key(*handle*, *password*, *address*)
Delete a key in the wallet.

Parameters

- **handle** (*str*) – wallet handle token
- **password** (*str*) – wallet password
- **address** (*str*) – base32 address of account to be deleted

Returns True if the account has been deleted

Return type bool

list_keys(*handle*)
List all keys in the wallet.

Parameters **handle** (*str*) – wallet handle token

Returns list of base32 addresses in the wallet

Return type str[]

sign_transaction(*handle*, *password*, *txn*, *signing_address=None*)
Sign a transaction.

Parameters

- **handle** (*str*) – wallet handle token
- **password** (*str*) – wallet password
- **txn** (Transaction) – transaction to be signed
- **signing_address** (*str, optional*) – sign the transaction with SK corresponding to base32 signing_address, if provided, rather than SK corresponding to sender

Returns signed transaction with signature of sender

Return type *SignedTransaction*

list_multisig(*handle*)
List all multisig accounts in the wallet.

Parameters **handle** (*str*) – wallet handle token

Returns list of base32 multisig account addresses

Return type str[]

import_multisig(*handle*, *multisig*)
Import a multisig account into the wallet.

Parameters

- **handle** (*str*) – wallet handle token
- **multisig** (Multisig) – multisig account to be imported

Returns base32 address of the imported multisig account

Return type str

export_multisig(*handle*, *address*)
Export a multisig account.

Parameters

- **handle** (*str*) – wallet token handle
- **address** (*str*) – base32 address of the multisig account

Returns multisig object corresponding to the address

Return type *Multisig*

delete_multisig(*handle*, *password*, *address*)
Delete a multisig account.

Parameters

- **handle** (*str*) – wallet handle token
- **password** (*str*) – wallet password
- **address** (*str*) – base32 address of the multisig account to delete

Returns True if the multisig account has been deleted

Return type bool

sign_multisig_transaction(*handle*, *password*, *public_key*, *mtx*)
Sign a multisig transaction for the given public key.

Parameters

- **handle** (*str*) – wallet handle token
- **password** (*str*) – wallet password
- **public_key** (*str*) – base32 address that is signing the transaction
- **mtx** (MultisigTransaction) – multisig transaction containing unsigned or partially signed multisig

Returns multisig transaction with added signature

Return type *MultisigTransaction*

10.1.9 logic

check_program(*program*, *args=None*)
Performs program checking for max length and cost

Parameters

- **program** (*bytes*) – compiled program
- **args** (*list[bytes]*) – args are not signed, but are checked by logic

Returns True on success

Return type bool

Raises InvalidProgram – on error

read_program(*program*, *args=None*)

check_int_const_block(*program*, *pc*)

read_int_const_block(*program*, *pc*)

check_byte_const_block(*program*, *pc*)

read_byte_const_block(*program*, *pc*)

parse_uvarint(*buf*)

address(*program*)
Return the address of the program.

Parameters **program** (*bytes*) – compiled program

Returns program address

Return type str

teal_sign(*private_key*, *data*, *contract_addr*)
Return the signature suitable for ed25519verify TEAL opcode

Parameters

- **private_key** (*str*) – private key to sign with
- **data** (*bytes*) – data to sign
- **contract_addr** (*str*) – program hash (contract address) to sign for

Returns signature

Return type bytes

teal_sign_from_program(*private_key*, *data*, *program*)
Return the signature suitable for ed25519verify TEAL opcode

Parameters

- **private_key** (*str*) – private key to sign with
- **data** (*bytes*) – data to sign
- **program** (*bytes*) – program to sign for

Returns signature

Return type bytes

10.1.10 mnemonic

from_master_derivation_key(*key*)
Return the mnemonic for the master derivation key.

Parameters **key** (*str*) – master derivation key in base64

Returns mnemonic

Return type str

to_master_derivation_key(*mnemonic*)
Return the master derivation key for the mnemonic.

Parameters **mnemonic** (*str*) – mnemonic of the master derivation key

Returns master derivation key in base64

Return type str

from_private_key(*key*)
Return the mnemonic for the private key.

Parameters **key** (*str*) – private key in base64

Returns mnemonic

Return type str

to_private_key(*mnemonic*)
Return the private key for the mnemonic.

Parameters **mnemonic** (*str*) – mnemonic of the private key

Returns private key in base64

Return type str

to_public_key(*mnemonic*)
Return the public key for the mnemonic.

Parameters **mnemonic** (*str*) – mnemonic of the public key

Returns public key in base32

Return type str

10.1.11 template

class Template
Bases: object

get_address()
Return the address of the contract.

get_program()

class Split(*owner: str*, *receiver_1: str*, *receiver_2: str*, *rat_1: int*, *rat_2: int*, *expiry_round: int*, *min_pay: int*, *max_fee: int*)
Bases: *algosdk.template.Template*

Split allows locking algos in an account which allows transfering to two predefined addresses in a specified ratio such that for the given ratn and ratd parameters we have:

first_recipient_amount * rat_2 == second_recipient_amount * rat_1

Split also has an expiry round, after which the owner can transfer back the funds.

Parameters

- **owner** (*str*) – an address that can receive the funds after the expiry round
- **receiver_1** (*str*) – first address to receive funds
- **receiver_2** (*str*) – second address to receive funds
- **rat_1** (*int*) – how much receiver_1 receives (proportionally)
- **rat_2** (*int*) – how much receiver_2 receives (proportionally)
- **expiry_round** (*int*) – the round on which the funds can be transferred back to owner
- **min_pay** (*int*) – the minimum number of microalgos that can be transferred from the account to receiver_1
- **max_fee** (*int*) – half the maximum fee that can be paid to the network by the account

get_program()
Return a byte array to be used in LogicSig.

static get_split_funds_transaction(*contract*, *amount: int*, *fee: int*, *first_valid*, *last_valid*, *gh*)
Return a group transactions array which transfers funds according to the contract's ratio. :param amount: total amount to be transferred :type amount: int :param fee: fee per byte :type fee: int :param first_valid: first round where the transactions are valid :type first_valid: int :param gh: genesis hash in base64 :type gh: str

Returns Transaction[]

class HTLC(*owner: str*, *receiver: str*, *hash_function: str*, *hash_image: str*, *expiry_round: int*, *max_fee: int*)
Bases: *algosdk.template.Template*

Hash Time Locked Contract allows a user to recieve the Algo prior to a deadline (in terms of a round) by proving knowledge of a special value or to forfeit the ability to claim, returning it to the payer.

This contract is usually used to perform cross-chained atomic swaps.

More formally, algos can be transfered under only two circumstances:

1. To receiver if hash_function(arg_0) = hash_value
2. To owner if txn.FirstValid > expiry_round

Parameters

- **owner** (*str*) – an address that can receive the asset after the expiry round
- **receiver** (*str*) – address to receive Algos
- **hash_function** (*str*) – the hash function to be used (must be either sha256 or keccak256)
- **hash_image** (*str*) – the hash image in base64
- **expiry_round** (*int*) – the round on which the assets can be transferred back to owner
- **max_fee** (*int*) – the maximum fee that can be paid to the network by the account

get_program()
Return a byte array to be used in LogicSig.

static get_transaction(*contract*, *preimage*, *first_valid*, *last_valid*, *gh*, *fee*)
Return a transaction which will release funds if a matching preimage is used.

Parameters

- **contract** (*bytes*) – the contract containing information, should be received from payer
- **preimage** (*str*) – the preimage of the hash in base64
- **first_valid** (*int*) – first valid round for the transactions
- **last_valid** (*int*) – last valid round for the transactions
- **gh** (*str*) – genesis hash in base64
- **fee** (*int*) – fee per byte

Returns

transaction to claim algos from contract account

Return type *LogicSigTransaction*

class DynamicFee(*receiver: str*, *amount: int*, *first_valid: int*, *last_valid: int = None*, *close_remainder_address: str = None*)
Bases: *algosdk.template.Template*

DynamicFee contract allows you to create a transaction without specifying the fee. The fee will be determined at the moment of transfer.

Parameters

- **receiver** (*str*) – address to receive the assets
- **amount** (*int*) – amount of assets to transfer
- **first_valid** (*int*) – first valid round for the transaction
- **last_valid** (*int, optional*) – last valid round for the transaction (defaults to first_valid + 1000)
- **close_remainder_address** (*str, optional*) – the address that recieves the remainder

get_program()
Return a byte array to be used in LogicSig.

static get_transactions(*txn*, *lsig*, *private_key*, *fee*)
Create and sign the secondary dynamic fee transaction, update transaction fields, and sign as the fee payer; return both transactions.

Parameters

- **txn** (Transaction) – main transaction from payer
- **lsig** (LogicSig) – signed logic received from payer
- **private_key** (*str*) – the secret key of the account that pays the fee in base64
- **fee** (*int*) – fee per byte, for both transactions

sign_dynamic_fee(*private_key*, *gh*)
Return the main transaction and signed logic needed to complete the transfer. These should be sent to the fee payer, who can use get_transactions() to update fields and create the auxiliary transaction.

Parameters

- **private_key** (*bytes*) – the secret key to sign the contract in base64

- **gh** (*str*) – genesis hash, in base64

class PeriodicPayment(*receiver: str*, *amount: int*, *withdrawing_window: int*, *period: int*, *max_fee: int*, *timeout: int*)
Bases: *algosdk.template.Template*

PeriodicPayment contract enables creating an account which allows the withdrawal of a fixed amount of assets every fixed number of rounds to a specific Algrorand Address. In addition, the contract allows to add timeout, after which the address can withdraw the rest of the assets.

Parameters

- **receiver** (*str*) – address to receive the assets
- **amount** (*int*) – amount of assets to transfer at every cycle
- **withdrawing_window** (*int*) – the number of blocks in which the user can withdraw the asset once the period start (must be < 1000)
- **period** (*int*) – how often the address can withdraw assets (in rounds)
- **fee** (*int*) – maximum fee per transaction
- **timeout** (*int*) – a round in which the receiver can withdraw the rest of the funds after

get_program()
Return a byte array to be used in LogicSig.

static get_withdrawal_transaction(*contract*, *first_valid*, *gh*, *fee*)
Return the withdrawal transaction to be sent to the network.

Parameters

- **contract** (*bytes*) – contract containing information, should be received from payer
- **first_valid** (*int*) – first round the transaction should be valid; this must be a multiple of self.period
- **gh** (*str*) – genesis hash in base64
- **fee** (*int*) – fee per byte

class LimitOrder(*owner: str*, *asset_id: int*, *ratn: int*, *ratd: int*, *expiry_round: int*, *max_fee: int*, *min_trade: int*)
Bases: *algosdk.template.Template*

Limit Order allows to trade Algos for other assets given a specific ratio; for N Algos, swap for Rate * N Assets.

Parameters

- **owner** (*str*) – an address that can receive the asset after the expiry round
- **asset_id** (*int*) – asset to be transfered
- **ratn** (*int*) – the numerator of the exchange rate
- **ratd** (*int*) – the denominator of the exchange rate
- **expiry_round** (*int*) – the round on which the assets can be transferred back to owner
- **max_fee** (*int*) – the maximum fee that can be paid to the network by the account
- **min_trade** (*int*) – the minimum amount (of Algos) to be traded away

get_program()
Return a byte array to be used in LogicSig.

static get_swap_assets_transactions(*contract: bytes*, *asset_amount: int*, *microalgo_amount: int*, *private_key: str*, *first_valid*, *last_valid*, *gh*, *fee*)

Return a group transactions array which transfer funds according to the contract's ratio.

Parameters

- **contract** (*bytes*) – the contract containing information, should be received from payer
- **asset_amount** (*int*) – the amount of assets to be sent
- **microalgo_amount** (*int*) – the amount of microalgos to be received
- **private_key** (*str*) – the secret key to sign the contract
- **first_valid** (*int*) – first valid round for the transactions
- **last_valid** (*int*) – last valid round for the transactions
- **gh** (*str*) – genesis hash in base64
- **fee** (*int*) – fee per byte

put_uvarint(*buf*, *x*)

inject(*orig*, *offsets*, *values*, *values_types*)

10.1.12 transaction

class Transaction(*sender*, *fee*, *first*, *last*, *note*, *gen*, *gh*, *lease*, *txn_type*, *rekey_to*)

Bases: object

Superclass for various transaction types.

get_txid()

Get the transaction's ID.

Returns transaction ID

Return type str

sign(*private_key*)

Sign the transaction with a private key.

Parameters **private_key** (*str*) – the private key of the signing account

Returns signed transaction with the signature

Return type *SignedTransaction*

raw_sign(*private_key*)

Sign the transaction.

Parameters **private_key** (*str*) – the private key of the signing account

Returns signature

Return type bytes

estimate_size()

dictify()

static undictify(*d*)

class PaymentTxn(*sender*, *fee*, *first*, *last*, *gh*, *receiver*, *amt*, *close_remainder_to=None*, *note=None*, *gen=None*, *flat_fee=False*, *lease=None*, *rekey_to=None*)

Bases: *algosdk.transaction.Transaction*

Represents a payment transaction.

Parameters

- **sender** (*str*) – address of the sender
- **fee** (*int*) – transaction fee (per byte if flat_fee is false)
- **first** (*int*) – first round for which the transaction is valid
- **last** (*int*) – last round for which the transaction is valid
- **gh** (*str*) – genesis_hash
- **receiver** (*str*) – address of the receiver
- **amt** (*int*) – amount in microAlgos to be sent
- **close_remainder_to** (*str, optional*) – if nonempty, account will be closed and remaining algos will be sent to this address
- **note** (*bytes, optional*) – arbitrary optional bytes
- **gen** (*str, optional*) – genesis_id
- **flat_fee** (*bool, optional*) – whether the specified fee is a flat fee
- **lease** (*byte[32], optional*) – specifies a lease, and no other transaction with the same sender and lease can be confirmed in this transaction's valid rounds
- **rekey_to** (*str, optional*) – additionally rekey the sender to this address

sender

Type str

fee

Type int

first_valid_round

Type int

last_valid_round

Type int

note

Type bytes

genesis_id

Type str

genesis_hash

Type str

group

Type bytes

receiver

Type str

amt

Type int

close_remainder_to

Type str

type

Type str

lease

Type byte[32]

rekey_to

Type str

dictify()

class KeyregTxn(*sender*, *fee*, *first*, *last*, *gh*, *votekey*, *selkey*, *votefst*, *votelst*, *votekd*, *note=None*, *gen=None*, *flat_fee=False*, *lease=None*, *rekey_to=None*)

Bases: *algosdk.transaction.Transaction*

Represents a key registration transaction.

Parameters

- **sender** (*str*) – address of sender
- **fee** (*int*) – transaction fee (per byte if flat_fee is false)
- **first** (*int*) – first round for which the transaction is valid
- **last** (*int*) – last round for which the transaction is valid
- **gh** (*str*) – genesis_hash
- **votekey** (*str*) – participation public key
- **selkey** (*str*) – VRF public key
- **votefst** (*int*) – first round to vote
- **votelst** (*int*) – last round to vote
- **votekd** (*int*) – vote key dilution
- **note** (*bytes, optional*) – arbitrary optional bytes
- **gen** (*str, optional*) – genesis_id
- **flat_fee** (*bool, optional*) – whether the specified fee is a flat fee
- **lease** (*byte[32], optional*) – specifies a lease, and no other transaction with the same sender and lease can be confirmed in this transaction's valid rounds
- **rekey_to** (*str, optional*) – additionally rekey the sender to this address

sender

Type str

fee

Type int

first_valid_round

Type int

last_valid_round

Type int

note

Type bytes

genesis_id

Type str

genesis_hash

Type str

group

Type bytes

votepk

Type str

selkey

Type str

votefst

Type int

votelst

Type int

votekd

Type int

type

Type str

lease

Type byte[32]

rekey_to

Type str

dictify()

class AssetConfigTxn(*sender*, *fee*, *first*, *last*, *gh*, *index=None*, *total=None*, *default_frozen=None*, *unit_name=None*, *asset_name=None*, *manager=None*, *reserve=None*, *freeze=None*, *clawback=None*, *url=None*, *metadata_hash=None*, *note=None*, *gen=None*, *flat_fee=False*, *lease=None*, *strict_empty_address_check=True*, *decimals=0*, *rekey_to=None*)

Bases: *algosdk.transaction.Transaction*

Represents a transaction for asset creation, reconfiguration, or destruction.

To create an asset, include the following: total, default_frozen, unit_name, asset_name, manager, reserve, freeze, clawback, url, metadata, decimals

To destroy an asset, include the following: index, strict_empty_address_check (set to False)

To update asset configuration, include the following: index, manager, reserve, freeze, clawback, strict_empty_address_check (optional)

Parameters

- **sender** (*str*) – address of the sender
- **fee** (*int*) – transaction fee (per byte if flat_fee is false)
- **first** (*int*) – first round for which the transaction is valid
- **last** (*int*) – last round for which the transaction is valid
- **gh** (*str*) – genesis_hash
- **index** (*int, optional*) – index of the asset
- **total** (*int, optional*) – total number of base units of this asset created
- **default_frozen** (*bool, optional*) – whether slots for this asset in user accounts are frozen by default
- **unit_name** (*str, optional*) – hint for the name of a unit of this asset
- **asset_name** (*str, optional*) – hint for the name of the asset
- **manager** (*str, optional*) – address allowed to change nonzero addresses for this asset
- **reserve** (*str, optional*) – account whose holdings of this asset should be reported as "not minted"
- **freeze** (*str, optional*) – account allowed to change frozen state of holdings of this asset
- **clawback** (*str, optional*) – account allowed take units of this asset from any account
- **url** (*str, optional*) – a URL where more information about the asset can be retrieved
- **metadata_hash** (*byte[32], optional*) – a commitment to some unspecified asset metadata (32 byte hash)
- **note** (*bytes, optional*) – arbitrary optional bytes
- **gen** (*str, optional*) – genesis_id
- **flat_fee** (*bool, optional*) – whether the specified fee is a flat fee
- **lease** (*byte[32], optional*) – specifies a lease, and no other transaction with the same sender and lease can be confirmed in this transaction's valid rounds
- **strict_empty_address_check** (*bool, optional*) – set this to False if you want to specify empty addresses. Otherwise, if this is left as True (the default), having empty addresses will raise an error, which will prevent accidentally removing admin access to assets or deleting the asset.
- **decimals** (*int, optional*) – number of digits to use for display after decimal. If set to 0, the asset is not divisible. If set to 1, the base unit of the asset is in tenths. Must be between 0 and 19, inclusive. Defaults to 0.
- **rekey_to** (*str, optional*) – additionally rekey the sender to this address

sender

Type str

fee

Type int

first_valid_round

Type int

last_valid_round

Type int

genesis_hash

Type str

index

Type int

total

Type int

default_frozen

Type bool

unit_name

Type str

asset_name

Type str

manager

Type str

reserve

Type str

freeze

Type str

clawback

Type str

url

Type str

metadata_hash

Type byte[32]

note

Type bytes

genesis_id

Type str

type

Type str

lease

Type byte[32]

decimals

Type int

rekey_to

Type str

dictify()

class AssetFreezeTxn(*sender*, *fee*, *first*, *last*, *gh*, *index*, *target*, *new_freeze_state*, *note=None*, *gen=None*, *flat_fee=False*, *lease=None*, *rekey_to=None*)

Bases: *algosdk.transaction.Transaction*

Represents a transaction for freezing or unfreezing an account's asset holdings. Must be issued by the asset's freeze manager.

Parameters

- **sender** (*str*) – address of the sender, who must be the asset's freeze manager
- **fee** (*int*) – transaction fee (per byte if flat_fee is false)
- **first** (*int*) – first round for which the transaction is valid
- **last** (*int*) – last round for which the transaction is valid
- **gh** (*str*) – genesis_hash
- **index** (*int*) – index of the asset
- **target** (*str*) – address having its assets frozen or unfrozen
- **new_freeze_state** (*bool*) – true if the assets should be frozen, false if they should be transferrable
- **note** (*bytes, optional*) – arbitrary optional bytes
- **gen** (*str, optional*) – genesis_id
- **flat_fee** (*bool, optional*) – whether the specified fee is a flat fee
- **lease** (*byte[32], optional*) – specifies a lease, and no other transaction with the same sender and lease can be confirmed in this transaction's valid rounds
- **rekey_to** (*str, optional*) – additionally rekey the sender to this address

sender

Type str

fee

Type int

first_valid_round

Type int

last_valid_round

Type int

genesis_hash

Type str

index

Type int

target

Type str

new_freeze_state

Type bool

note

Type bytes

genesis_id

Type str

type

Type str

lease

Type byte[32]

rekey_to

Type str

dictify()

class AssetTransferTxn(*sender*, *fee*, *first*, *last*, *gh*, *receiver*, *amt*, *index*, *close_assets_to=None*, *revocation_target=None*, *note=None*, *gen=None*, *flat_fee=False*, *lease=None*, *rekey_to=None*)

Bases: *algosdk.transaction.Transaction*

Represents a transaction for asset transfer. To begin accepting an asset, supply the same address as both sender and receiver, and set amount to 0. To revoke an asset, set revocation_target, and issue the transaction from the asset's revocation manager account.

Parameters

- **sender** (*str*) – address of the sender
- **fee** (*int*) – transaction fee (per byte if flat_fee is false)
- **first** (*int*) – first round for which the transaction is valid
- **last** (*int*) – last round for which the transaction is valid
- **gh** (*str*) – genesis_hash
- **receiver** (*str*) – address of the receiver
- **amt** (*int*) – amount of asset base units to send
- **index** (*int*) – index of the asset
- **close_assets_to** (*string, optional*) – send all of sender's remaining assets, after paying *amt* to receiver, to this address
- **revocation_target** (*string, optional*) – send assets from this address, rather than the sender's address (can only be used by an asset's revocation manager, also known as clawback)
- **note** (*bytes, optional*) – arbitrary optional bytes

- **gen** (*str, optional*) – genesis_id
- **flat_fee** (*bool, optional*) – whether the specified fee is a flat fee
- **lease** (*byte[32], optional*) – specifies a lease, and no other transaction with the same sender and lease can be confirmed in this transaction's valid rounds
- **rekey_to** (*str, optional*) – additionally rekey the sender to this address

sender

Type str

fee

Type int

first_valid_round

Type int

last_valid_round

Type int

genesis_hash

Type str

index

Type int

amount

Type int

receiver

Type string

close_assets_to

Type string

revocation_target

Type string

note

Type bytes

genesis_id

Type str

type

Type str

lease

Type byte[32]

rekey_to

Type str

dictify()

class SignedTransaction(*transaction*, *signature*, *authorizing_address=None*)
Bases: object

Represents a signed transaction.

Parameters

- **transaction** (Transaction) – transaction that was signed
- **signature** (*str*) – signature of a single address
- **authorizing_address** (*str, optional*) – the address authorizing the signed transaction, if different from sender

transaction

Type *Transaction*

signature

Type str

authorizing_address

Type str

dictify()

static undictify(*d*)

class MultisigTransaction(*transaction*, *multisig*)
Bases: object

Represents a signed transaction.

Parameters

- **transaction** (Transaction) – transaction that was signed
- **multisig** (Multisig) – multisig account and signatures

transaction

Type *Transaction*

multisig

Type *Multisig*

sign(*private_key*)
Sign the multisig transaction.

Parameters **private_key** (*str*) – private key of signing account

Note: A new signature will replace the old if there is already a signature for the address. To sign another transaction, you can either overwrite the signatures in the current Multisig, or you can use Multisig.get_multisig_account() to get a new multisig object with the same addresses.

dictify()

static undictify(*d*)

static merge(*part_stxs*)
Merge partially signed multisig transactions.

Parameters **part_stxs** (*MultisigTransaction[]*) – list of partially signed multisig transactions

Returns multisig transaction containing signatures

Return type *MultisigTransaction*

Note: Only use this if you are given two partially signed multisig transactions. To append a signature to a multisig transaction, just use MultisigTransaction.sign()

class Multisig(*version*, *threshold*, *addresses*)

Bases: object

Represents a multisig account and signatures.

Parameters

- **version** (*int*) – currently, the version is 1
- **threshold** (*int*) – how many signatures are necessary
- **addresses** (*str[]*) – addresses in the multisig account

version

Type int

threshold

Type int

subsigs

Type *MultisigSubsig*[]

validate()

Check if the multisig account is valid.

address()

Return the multisig account address.

verify(*message*)

Verify that the multisig is valid for the message.

dictify()

json_dictify()

static undictify(*d*)

get_multisig_account()

Return a Multisig object without signatures.

get_public_keys()

Return the base32 encoded addresses for the multisig account.

class MultisigSubsig(*public_key*, *signature=None*)

Bases: object

public_key

Type bytes

signature

Type bytes

dictify()

json_dictify()

static undictify(*d*)

class LogicSig(*program*, *args=None*)
Bases: object

Represents a logic signature.

Parameters

- **logic** (*bytes*) – compiled program
- **args** (*list[bytes]*) – args are not signed, but are checked by logic

logic

Type bytes

sig

Type bytes

msig

Type *Multisig*

args

Type list[bytes]

dictify()

static undictify(*d*)

verify(*public_key*)
Verifies LogicSig against the transaction's sender address

Parameters **public_key** (*bytes*) – sender address

Returns true if the signature is valid (the sender address matches the logic hash or the signature is valid against the sender address), false otherwise

Return type bool

address()
Compute hash of the logic sig program (that is the same as escrow account address) as string address

Returns program address

Return type str

static sign_program(*program*, *private_key*)

static single_sig_multisig(*program*, *private_key*, *multisig*)

sign(*private_key*, *multisig=None*)
Creates signature (if no pk provided) or multi signature

Parameters

- **private_key** (*str*) – private key of signing account
- **multisig** (Multisig) – optional multisig account without signatures to sign with

Raises InvalidSecretKeyError – if no matching private key in multisig object

append_to_multisig(*private_key*)
Appends a signature to multi signature

Parameters **private_key** (*str*) – private key of signing account

Raises InvalidSecretKeyError – if no matching private key in multisig object

class LogicSigTransaction(*transaction*, *lsig*)
Bases: object

Represents a logic signed transaction.

Parameters

- **transaction** (Transaction) –
- **lsig** (LogicSig) –

transaction

Type *Transaction*

lsig

Type *LogicSig*

verify()
Verify LogicSig against the transaction

Returns true if the signature is valid (the sender address matches the logic hash or the signature is valid against the sender address), false otherwise

Return type bool

dictify()

static undictify(*d*)

write_to_file(*objs*, *path*, *overwrite=True*)
Write signed or unsigned transactions to a file.

Parameters

- **txns** (Transaction[], SignedTransaction[], or MultisigTransaction[]) – can be a mix of the three
- **path** (*str*) – file to write to
- **overwrite** (*bool*) – whether or not to overwrite what's already in the file; if False, transactions will be appended to the file

Returns true if the transactions have been written to the file

Return type bool

retrieve_from_file(*path*)
Retrieve encoded objects from a file.

Parameters **path** (*str*) – file to read from

Returns list of objects

Return type Object[]

class TxGroup(*txns*)
Bases: object

dictify()

static undictify(*d*)

calculate_group_id(*txns*)
Calculate group id for a given list of unsigned transactions

Parameters **txns** (*list*) – list of unsigned transactions

Returns checksum value representing the group id

Return type bytes

assign_group_id(*txns*, *address=None*)
Assign group id to a given list of unsigned transactions.

Parameters

- **txns** (*list*) – list of unsigned transactions
- **address** (*str*) – optional sender address specifying which transaction to return

Returns list of unsigned transactions with group property set

Return type txns (list)

10.1.13 util

microalgos_to_algos(*microalgos*)
Convert microalgos to algos.

Parameters **microalgos** (*int*) – how many microalgos

Returns how many algos

Return type int or decimal

algos_to_microalgos(*algos*)
Convert algos to microalgos.

Parameters **algos** (*int or decimal*) – how many algos

Returns how many microalgos

Return type int

sign_bytes(*to_sign*, *private_key*)
Sign arbitrary bytes after prepending with "MX" for domain separation.

Parameters **to_sign** (*bytes*) – bytes to sign

Returns base64 signature

Return type str

verify_bytes(*message*, *signature*, *public_key*)
Verify the signature of a message that was prepended with "MX" for domain separation.

Parameters

- **message** (*bytes*) – message that was signed, without prefix
- **signature** (*str*) – base64 signature
- **public_key** (*str*) – base32 address

Returns whether or not the signature is valid

Return type bool

10.1.14 v2client

10.1.14.1 v2client.algod

class AlgodClient(*algod_token*, *algod_address*, *headers=None*)

Bases: object

Client class for algod. Handles all algod requests.

Parameters

- **algod_token** (*str*) – algod API token
- **algod_address** (*str*) – algod address
- **headers** (*dict, optional*) – extra header name/value for all requests

algod_token

Type str

algod_address

Type str

headers

Type dict

algod_request(*method*, *requrl*, *params=None*, *data=None*, *headers=None*, *response_format='json'*)

Execute a given request.

Parameters

- **method** (*str*) – request method
- **requrl** (*str*) – url for the request
- **params** (*dict, optional*) – parameters for the request
- **data** (*dict, optional*) – data in the body of the request
- **headers** (*dict, optional*) – additional header for request

Returns loaded from json response body

Return type dict

account_info(*address*, ***kwargs*)

Return account information.

Parameters **address** (*str*) – account public key

pending_transactions_by_address(*address*, *limit=0*, *response_format='json'*, ***kwargs*)

Get the list of pending transactions by address, sorted by priority, in decreasing order, truncated at the end at MAX. If MAX = 0, returns all pending transactions.

Parameters

- **address** (*str*) – account public key
- **limit** (*int, optional*) – maximum number of transactions to return
- **response_format** (*str*) – the format in which the response is returned: either "json" or "msgpack"

block_info(*block*, *response_format='json'*, ***kwargs*)
Get the block for the given round.

Parameters

- **block** (*int*) – block number
- **response_format** (*str*) – the format in which the response is returned: either "json" or "msgpack"

ledger_supply(***kwargs*)
Return supply details for node's ledger.

status(***kwargs*)
Return node status.

status_after_block(*block_num*, ***kwargs*)
Return node status immediately after blockNum.

Parameters **block_num** – block number

send_transaction(*txn*, ***kwargs*)
Broadcast a signed transaction object to the network.

Parameters

- **txn** (*SignedTransaction or MultisigTransaction*) – transaction to send
- **request_header** (*dict, optional*) – additional header for request

Returns transaction ID

Return type str

send_raw_transaction(*txn*, ***kwargs*)
Broadcast a signed transaction to the network.

Parameters

- **txn** (*str*) – transaction to send, encoded in base64
- **request_header** (*dict, optional*) – additional header for request

Returns transaction ID

Return type str

pending_transactions(*max_txns=0*, *response_format='json'*, ***kwargs*)
Return pending transactions.

Parameters

- **max_txns** (*int*) – maximum number of transactions to return; if max_txns is 0, return all pending transactions
- **response_format** (*str*) – the format in which the response is returned: either "json" or "msgpack"

pending_transaction_info(*transaction_id*, *response_format='json'*, ***kwargs*)
Return transaction information for a pending transaction.

Parameters

- **transaction_id** (*str*) – transaction ID
- **response_format** (*str*) – the format in which the response is returned: either "json" or "msgpack"

health(***kwargs*)
Return null if the node is running.

versions(***kwargs*)
Return algod versions.

send_transactions(*txns*, ***kwargs*)
Broadcast list of a signed transaction objects to the network.

Parameters

- **txns** (*SignedTransaction[] or MultisigTransaction[]*) – transactions to send
- **request_header** (*dict, optional*) – additional header for request

Returns first transaction ID

Return type str

suggested_params(***kwargs*)
Return suggested transaction parameters.

compile(*source*, ***kwargs*)
Compile TEAL source with remote algod.

Parameters

- **source** (*str*) – source to be compiled
- **request_header** (*dict, optional*) – additional header for request

Returns compilation result

Return type bytes

dryrun(*drr*, ***kwargs*)
Dryrun with remote algod.

Parameters

- **drr** (*obj*) – dryrun request object
- **request_header** (*dict, optional*) – additional header for request

Returns compilation result

Return type bytes

10.1.14.2 v2client.indexer

class IndexerClient(*indexer_token*, *indexer_address*, *headers=None*)
Bases: object

Client class for indexer. Handles all indexer requests.

Parameters

- **indexer_token** (*str*) – indexer API token
- **indexer_address** (*str*) – indexer address
- **headers** (*dict, optional*) – extra header name/value for all requests

indexer_token

Type str

indexer_address

Type str

headers

Type dict

indexer_request(*method*, *requrl*, *params=None*, *data=None*, *headers=None*)
Execute a given request.

Parameters

- **method** (*str*) – request method
- **requrl** (*str*) – url for the request
- **params** (*dict, optional*) – parameters for the request
- **data** (*dict, optional*) – data in the body of the request
- **headers** (*dict, optional*) – additional header for request

Returns loaded from json response body

Return type dict

health(***kwargs*)
Return 200 and a simple status message if the node is running.

accounts(*asset_id=None*, *limit=None*, *next_page=None*, *min_balance=None*, *max_balance=None*, *block=None*, *auth_addr=None*, *application_id=None*, ***kwargs*)
Return accounts that match the search; microalgos are the default currency unless asset_id is specified, in which case the asset will be used.

Parameters

- **asset_id** (*int, optional*) – include accounts holding this asset
- **limit** (*int, optional*) – maximum number of results to return
- **next_page** (*str, optional*) – the next page of results; use the next token provided by the previous results
- **min_balance** (*int, optional*) – results should have an amount greater than this value
- **max_balance** (*int, optional*) – results should have an amount less
- **block** (*int, optional*) – include results for the specified round; for performance reasons, this parameter may be disabled on some configurations
- **auth_addr** (*str, optional*) – Include accounts configured to use this spending key.
- **application_id** (*int, optional*) – results should filter on this application

asset_balances(*asset_id*, *limit=None*, *next_page=None*, *min_balance=None*, *max_balance=None*, *block=None*, ***kwargs*)
Return accounts that hold the asset; microalgos are the default currency unless asset_id is specified, in which case the asset will be used.

Parameters

- **asset_id** (*int*) – include accounts holding this asset
- **limit** (*int, optional*) – maximum number of results to return

- **next_page** (*str, optional*) – the next page of results; use the next token provided by the previous results
- **min_balance** (*int, optional*) – results should have an amount greater than this value
- **max_balance** (*int, optional*) – results should have an amount less
- **block** (*int, optional*) – include results for the specified round; for performance reasons, this parameter may be disabled on some configurations

block_info(*block*, ***kwargs*)
Get the block for the given round.

Parameters **block** (*int*) – block number

account_info(*address*, *block=None*, ***kwargs*)
Return account information.

Parameters

- **address** (*str*) – account public key
- **block** (*int, optional*) – use results from the specified round

search_transactions(*limit=None*, *next_page=None*, *note_prefix=None*, *txn_type=None*, *sig_type=None*, *txid=None*, *block=None*, *min_round=None*, *max_round=None*, *asset_id=None*, *start_time=None*, *end_time=None*, *min_amount=None*, *max_amount=None*, *address=None*, *address_role=None*, *exclude_close_to=False*, *application_id=None*, *rekey_to=False*, ***kwargs*)
Return a list of transactions satisfying the conditions.

Parameters

- **limit** (*int, optional*) – maximum number of results to return
- **next_page** (*str, optional*) – the next page of results; use the next token provided by the previous results
- **note_prefix** (*bytes, optional*) – specifies a prefix which must be contained in the note field
- **txn_type** (*str, optional*) – type of transaction; one of “pay”, “keyreg”, “acfg”, “axfer”, “afrz”
- **sig_type** (*str, optional*) – type of signature; one of “sig”, “msig”, “lsig”
- **txid** (*str, optional*) – lookup a specific transaction by ID
- **block** (*int, optional*) – include results for the specified round
- **min_round** (*int, optional*) – include results at or after the specified round
- **max_round** (*int, optional*) – include results at or before the specified round
- **asset_id** (*int, optional*) – include transactions for the specified asset
- **end_time** (*str, optional*) – include results before the given time; must be an RFC 3339 formatted string
- **start_time** (*str, optional*) – include results after the given time; must be an RFC 3339 formatted string

- **min_amount** (*int, optional*) – results should have an amount greater than this value; microalgos are the default currency unless an asset-id is provided, in which case the asset will be used
- **max_amount** (*int, optional*) – results should have an amount less than this value, microalgos are the default currency unless an asset-id is provided, in which case the asset will be used
- **address** (*str, optional*) – only include transactions with this address in one of the transaction fields
- **address_role** (*str, optional*) – one of "sender" or "receiver"; combine with the address parameter to define what type of address to search for
- **exclude_close_to** (*bool, optional*) – combine with address and address_role parameters to define what type of address to search for; the close to fields are normally treated as a receiver, if you would like to exclude them set this parameter to true
- **application_id** (*int, optional*) – filter for transactions pertaining to an application
- **rekey_to** (*bool, optional*) – rekey-to field.

search_transactions_by_address(*address*, *limit=None*, *next_page=None*, *note_prefix=None*, *txn_type=None*, *sig_type=None*, *txid=None*, *block=None*, *min_round=None*, *max_round=None*, *asset_id=None*, *start_time=None*, *end_time=None*, *min_amount=None*, *max_amount=None*, *rekey_to=False*, ***kwargs*)

Return a list of transactions satisfying the conditions for the address.

Parameters

- **address** (*str*) – only include transactions with this address in one of the transaction fields
- **limit** (*int, optional*) – maximum number of results to return
- **next_page** (*str, optional*) – the next page of results; use the next token provided by the previous results
- **note_prefix** (*bytes, optional*) – specifies a prefix which must be contained in the note field
- **txn_type** (*str, optional*) – type of transaction; one of "pay", "keyreg", "acfg", "axfer", "afrz"
- **sig_type** (*str, optional*) – type of signature; one of "sig", "msig", "lsig"
- **txid** (*str, optional*) – lookup a specific transaction by ID
- **block** (*int, optional*) – include results for the specified round
- **min_round** (*int, optional*) – include results at or after the specified round
- **max_round** (*int, optional*) – include results at or before the specified round
- **asset_id** (*int, optional*) – include transactions for the specified asset
- **end_time** (*str, optional*) – include results before the given time; must be an RFC 3339 formatted string
- **start_time** (*str, optional*) – include results after the given time; must be an RFC 3339 formatted string

- **min_amount** (*int, optional*) – results should have an amount greater than this value; microalgos are the default currency unless an asset-id is provided, in which case the asset will be used
- **max_amount** (*int, optional*) – results should have an amount less than this value, microalgos are the default currency unless an asset-id is provided, in which case the asset will be used
- **rekey_to** (*bool, optional*) – rekey-to field.

search_asset_transactions(*asset_id*, *limit=None*, *next_page=None*, *note_prefix=None*, *txn_type=None*, *sig_type=None*, *txid=None*, *block=None*, *min_round=None*, *max_round=None*, *address=None*, *start_time=None*, *end_time=None*, *min_amount=None*, *max_amount=None*, *address_role=None*, *exclude_close_to=False*, *rekey_to=False*, ***kwargs*)

Return a list of transactions satisfying the conditions for the address.

Parameters

- **asset_id** (*int*) – include transactions for the specified asset
- **limit** (*int, optional*) – maximum number of results to return
- **next_page** (*str, optional*) – the next page of results; use the next token provided by the previous results
- **note_prefix** (*bytes, optional*) – specifies a prefix which must be contained in the note field
- **txn_type** (*str, optional*) – type of transaction; one of “pay”, “keyreg”, “acfg”, “axfer”, “afrz”
- **sig_type** (*str, optional*) – type of signature; one of “sig”, “msig”, “lsig”
- **txid** (*str, optional*) – lookup a specific transaction by ID
- **block** (*int, optional*) – include results for the specified round
- **min_round** (*int, optional*) – include results at or after the specified round
- **max_round** (*int, optional*) – include results at or before the specified round
- **address** (*str, optional*) – only include transactions with this address in one of the transaction fields
- **end_time** (*str, optional*) – include results before the given time; must be an RFC 3339 formatted string
- **start_time** (*str, optional*) – include results after the given time; must be an RFC 3339 formatted string
- **min_amount** (*int, optional*) – results should have an amount greater than this value; microalgos are the default currency unless an asset-id is provided, in which case the asset will be used
- **max_amount** (*int, optional*) – results should have an amount less than this value, microalgos are the default currency unless an asset-id is provided, in which case the asset will be used
- **address_role** (*str, optional*) – one of “sender” or “receiver”; combine with the address parameter to define what type of address to search for

- **exclude_close_to** (*bool, optional*) – combine with address and address_role parameters to define what type of address to search for; the close to fields are normally treated as a receiver, if you would like to exclude them set this parameter to true
- **rekey_to** (*bool, optional*) – rekey-to field.

search_assets(*limit=None*, *next_page=None*, *creator=None*, *name=None*, *unit=None*, *asset_id=None*, ***kwargs*)
Return assets that satisfy the conditions.

Parameters

- **limit** (*int, optional*) – maximum number of results to return
- **next_page** (*str, optional*) – the next page of results; use the next token provided by the previous results
- **creator** (*str, optional*) – filter just assets with the given creator address
- **name** (*str, optional*) – filter just assets with the given name
- **unit** (*str, optional*) – filter just assets with the given unit
- **asset_id** (*int, optional*) – return only the asset with this ID

asset_info(*asset_id*, ***kwargs*)
Return asset information.

Parameters **asset_id** (*int*) – asset index

applications(*application_id*, *round=None*, ***kwargs*)
Return applications that satisfy the conditions.

Parameters

- **application_id** (*int*) – application index
- **round** (*int, optional*) – restrict search to passed round

search_applications(*application_id=None*, *round=None*, *limit=None*, *next_page=None*, ***kwargs*)
Return applications that satisfy the conditions.

Parameters

- **application_id** (*int, optional*) – restrict search to application index
- **round** (*int, optional*) – restrict search to passed round
- **limit** (*int, optional*) – restrict number of results to limit
- **next_page** (*string, optional*) – used for pagination

10.1.15 wallet

class Wallet(*wallet_name*, *wallet_pswd*, *kmd_client*, *driver_name='sqlite'*, *mdk=None*)
Bases: object

Represents a wallet.

Parameters

- **wallet_name** (*str*) – wallet name
- **wallet_pswd** (*str*) – wallet password

- **kmd_client** (KMDClient) – a KMDClient to handle wallet requests
- **mdk** (*str, optional*) – master derivation key if recovering wallet

Note: When initializing, if the wallet doesn't already exist, it will be created.

name

Type str

pswd

Type str

kcl

Type *KMDClient*

id

Type str

handle

Type str

info()
Get wallet information.

Returns dictionary containing wallet handle and wallet information

Return type dict

list_keys()
List all keys in the wallet.

Returns list of base32 addresses in the wallet

Return type str[]

rename(*new_name*)
Rename the wallet.

Parameters **new_name** (*str*) – new name for the wallet

Returns dictionary containing wallet information

Return type dict

get_mnemonic()
Get recovery phrase mnemonic for the wallet.

Returns mnemonic converted from the wallet's master derivation key

Return type str

export_master_derivation_key()
Get the wallet's master derivation key.

Returns master derivation key

Return type str

import_key(*private_key*)
Import an account into a wallet.

Parameters **private_key** (*str*) – private key of account to be imported

Returns base32 address of the account

Return type str

export_key(*address*)
Return an account private key.

Parameters **address** (*str*) – base32 address of the account

Returns private key

Return type str

generate_key(*display_mnemonic=True*)
Generate a key in the wallet.

Parameters **display_mnemonic** (*bool, optional*) – whether or not the mnemonic should be displayed

Returns base32 address of the generated account

Return type str

delete_key(*address*)
Delete a key in the wallet.

Parameters **address** (*str*) – base32 address of account to be deleted

Returns True if the account has been deleted

Return type bool

sign_transaction(*txn*)
Sign a transaction.

Parameters **txn** (Transaction) – transaction to be signed

Returns signed transaction with signature of sender

Return type *SignedTransaction*

list_multisig()
List all multisig accounts in the wallet.

Returns list of base32 multisig account addresses

Return type str[]

import_multisig(*multisig*)
Import a multisig account into the wallet.

Parameters **multisig** (Multisig) – multisig account to be imported

Returns base32 address of the imported multisig account

Return type str

export_multisig(*address*)
Export a multisig account.

Parameters **address** (*str*) – base32 address of the multisig account

Returns multisig object corresponding to the address

Return type *Multisig*

delete_multisig(*address*)
Delete a multisig account.

Parameters **address** (*str*) – base32 address of the multisig account to delete

Returns True if the multisig account has been deleted

Return type bool

sign_multisig_transaction(*public_key*, *mtx*)
Sign a multisig transaction for the given public key.

Parameters

- **public_key** (*str*) – base32 address that is signing the transaction
- **mtx** (MultisigTransaction) – object containing unsigned or partially signed multisig

Returns multisig transaction with added signature

Return type *MultisigTransaction*

automate_handle()
Get a new handle or renews the current one.

Returns True if a handle is active

Return type bool

init_handle()
Get a new handle.

Returns True if a handle is active

Return type bool

renew_handle()
Renew the current handle.

Returns dictionary containing wallet handle and wallet information

Return type dict

release_handle()
Deactivate the current handle.

Returns True if the handle has been deactivated

Return type bool

10.1.16 wordlist

word_list_raw()
Return the wordlist used for mnemonics.

Python Module Index

a

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W