
algosdk

Algorand

May 02, 2022

CONTENTS

1	Installation	3
2	SDK Development	5
3	Quick start	7
4	Node setup	9
5	Running examples/example.py	11
6	Documentation	13
7	License	15
8	Modules	17
8.1	algosdk	17
8.1.1	abi	17
8.1.1.1	address_type	17
8.1.1.2	array_dynamic_type	18
8.1.1.3	array_static_type	18
8.1.1.4	base_type	19
8.1.1.5	bool_type	20
8.1.1.6	byte_type	20
8.1.1.7	contract	21
8.1.1.8	interface	21
8.1.1.9	method	22
8.1.1.10	reference	23
8.1.1.11	string_type	23
8.1.1.12	transaction	23
8.1.1.13	tuple_type	24
8.1.1.14	ufixed_type	25
8.1.1.15	uint_type	25
8.1.2	account	26
8.1.3	algod	26
8.1.4	atomic_transaction_composer	29
8.1.5	auction	34
8.1.6	constants	35
8.1.7	encoding	38
8.1.8	error	39
8.1.9	future	41
8.1.9.1	future.template	41

8.1.9.2	future.transaction	45
8.1.10	kmd	71
8.1.11	logic	75
8.1.12	mnemonic	77
8.1.13	template	77
8.1.14	transaction	81
8.1.15	util	94
8.1.16	v2client	95
8.1.16.1	v2client.algod	95
8.1.16.2	v2client.indexer	98
8.1.17	wallet	104
8.1.18	wordlist	107
Python Module Index		109
Index		111

A python library for interacting with the Algorand network.

INSTALLATION

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

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

SDK DEVELOPMENT

Install dependencies

- `pip install -r requirements.txt`

Run tests

- `make docker-test`

Format code:

- `black .`

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.")
```


NODE SETUP

Follow the instructions in Algorand's [developer resources](#) to install a node on your computer.

RUNNING EXAMPLES/EXAMPLE.PY

Before running `example.py`, start kmd on a private network or testnet node:

```
./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 `tokens.py`, either update the tokens and addresses, or provide a path to the data directory.

You're now ready to run `example.py`!

DOCUMENTATION

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

LICENSE

py-algorand-sdk is licensed under an MIT license. See the [LICENSE](#) file for details.

MODULES

8.1 algosdk

8.1.1 abi

8.1.1.1 address_type

class AddressType

Bases: *algosdk.abi.base_type.ABIStructType*

Represents an Address ABI Type for encoding.

byte_len() → int

Return the length in bytes of the ABI type.

is_dynamic() → bool

Return whether the ABI type is dynamic.

encode(*value: Union[str, bytes]*) → bytes

Encode an address string or a 32-byte public key into a Address ABI bytestring.

Parameters

- **value** (*str* / *bytes*) – value to be encoded. It can be either a base32
- **string or a 32-byte public key.** (*address*) –

Returns encoded bytes of the address

Return type bytes

decode(*bytestring: Union[bytearray, bytes]*) → str

Decodes a bytestring to a base32 encoded address string.

Parameters **bytestring** (*bytes* / *bytearray*) – bytestring to be decoded

Returns base32 encoded address from the encoded bytestring

Return type str

8.1.1.2 array_dynamic_type

class `ArrayDynamicType`(*arg_type*: `algosdk.abi.base_type.ABType`)

Bases: `algosdk.abi.base_type.ABType`

Represents a ArrayDynamic ABI Type for encoding.

Parameters `child_type` (`ABType`) – the type of the dynamic array.

child_type

Type `ABType`

byte_len() → NoReturn

Return the length in bytes of the ABI type.

is_dynamic() → bool

Return whether the ABI type is dynamic.

encode(*value_array*: `Union[List[Any], bytes, bytearray]`) → bytes

Encodes a list of values into a ArrayDynamic ABI bytestring.

Parameters

- **value_array** (`list` | `bytes` | `bytearray`) – list of values to be encoded.
- **the child types are ByteType** (*If*) –
- **bytes or bytearray can be** (*then*) –
- **in to be encoded as well.** (*passed*) –

Returns encoded bytes of the dynamic array

Return type bytes

decode(*array_bytes*: `Union[bytes, bytearray]`) → list

Decodes a bytestring to a dynamic list.

Parameters **array_bytes** (`bytes` | `bytearray`) – bytestring to be decoded

Returns values from the encoded bytestring

Return type list

8.1.1.3 array_static_type

class `ArrayStaticType`(*arg_type*: `algosdk.abi.base_type.ABType`, *array_len*: `int`)

Bases: `algosdk.abi.base_type.ABType`

Represents a ArrayStatic ABI Type for encoding.

Parameters

- **child_type** (`ABType`) – the type of the child_types array.
- **array_len** (`int`) – length of the static array.

child_type

Type `ABType`

static_length

Type `int`

byte_len() → int

Return the length in bytes of the ABI type.

is_dynamic() → bool

Return whether the ABI type is dynamic.

encode(*value_array: Union[List[Any], bytes, bytearray]*) → bytes

Encodes a list of values into a ArrayStatic ABI bytestring.

Parameters

- **value_array** (*list | bytes | bytearray*) – list of values to be encoded.
- **number of elements must match the predefined length of array.** (*The*) –
- **the child types are ByteType** (*If*) –
- **bytes or bytearray can be** (*then*) –
- **in to be encoded as well.** (*passed*) –

Returns encoded bytes of the static array

Return type bytes

decode(*array_bytes: Union[bytes, bytearray]*) → list

Decodes a bytestring to a static list.

Parameters **array_bytes** (*bytes | bytearray*) – bytestring to be decoded

Returns values from the encoded bytestring

Return type list

8.1.1.4 base_type

class **ABIType**

Bases: `abc.ABC`

Represents an ABI Type for encoding.

abstract **is_dynamic()** → bool

Return whether the ABI type is dynamic.

abstract **byte_len()** → int

Return the length in bytes of the ABI type.

abstract **encode**(*value: Any*) → bytes

Serialize the ABI value into a byte string using ABI encoding rules.

abstract **decode**(*bytestring: bytes*) → Any

Deserialize the ABI type and value from a byte string using ABI encoding rules.

static **from_string**(*s: str*) → [algosdk.abi.base_type.ABIType](#)

Convert a valid ABI string to a corresponding ABI type.

8.1.1.5 bool_type

class BoolType

Bases: *algosdk.abi.base_type.ABType*

Represents a Bool ABI Type for encoding.

byte_len() → int

Return the length in bytes of the ABI type.

is_dynamic() → bool

Return whether the ABI type is dynamic.

encode(value: bool) → bytes

Encode a boolean value

Parameters value (bool) – value to be encoded

Returns encoded bytes (“0x80” if True, “0x00” if False) of the boolean

Return type bytes

decode(bytestring: Union[bytes, bytearray]) → bool

Decodes a bytestring to a single boolean.

Parameters bytestring (bytes | bytearray) – bytestring to be decoded that contains a single boolean, i.e. “0x80” or “0x00”

Returns boolean from the encoded bytestring

Return type bool

8.1.1.6 byte_type

class ByteType

Bases: *algosdk.abi.base_type.ABType*

Represents a Byte ABI Type for encoding.

byte_len() → int

Return the length in bytes of the ABI type.

is_dynamic() → bool

Return whether the ABI type is dynamic.

encode(value: int) → bytes

Encode a single byte or a uint8

Parameters value (int) – value to be encoded

Returns encoded bytes of the uint8

Return type bytes

decode(bytestring: Union[bytes, bytearray]) → bytes

Decodes a bytestring to a single byte.

Parameters bytestring (bytes | bytearray) – bytestring to be decoded

Returns byte of the encoded bytestring

Return type bytes

8.1.1.7 contract

```
class Contract(name: str, methods: List[algosdk.abi.method.Method], desc: Optional[str] = None, networks:
    Optional[Dict[str, algosdk.abi.contract.NetworkInfo]] = None)
```

Bases: object

Represents a ABI contract description.

Parameters

- **name** (*string*) – name of the contract
- **methods** (*list*) – list of Method objects
- **desc** (*string*, *optional*) – description of the contract
- **networks** (*dict*, *optional*) – information about the contract in a particular network, such as an app-id.

```
static from_json(resp: Union[str, bytes, bytearray]) → algosdk.abi.contract.Contract
```

```
dictify() → dict
```

```
static undictify(d: dict) → algosdk.abi.contract.Contract
```

```
class NetworkInfo(app_id: int)
```

Bases: object

Represents network information.

Parameters **app_id** (*int*) – application ID on a particular network

```
dictify() → dict
```

```
static undictify(d: dict) → algosdk.abi.contract.NetworkInfo
```

8.1.1.8 interface

```
class Interface(name: str, methods: List[algosdk.abi.method.Method], desc: Optional[str] = None)
```

Bases: object

Represents a ABI interface description.

Parameters

- **name** (*string*) – name of the interface
- **methods** (*list*) – list of Method objects
- **desc** (*string*, *optional*) – description of the interface

```
static from_json(resp: Union[str, bytes, bytearray]) → algosdk.abi.interface.Interface
```

```
dictify() → dict
```

```
static undictify(d: dict) → algosdk.abi.interface.Interface
```

8.1.1.9 method

class Method(*name: str, args: List[algosdk.abi.method.Argument], returns: algosdk.abi.method.Returns, desc: Optional[str] = None*)

Bases: object

Represents a ABI method description.

Parameters

- **name** (*string*) – name of the method
- **args** (*list*) – list of Argument objects with type, name, and optional
- **description** –
- **returns** (*Returns*) – a Returns object with a type and optional description
- **desc** (*string, optional*) – optional description of the method

get_signature() → str

get_selector() → bytes

Returns the ABI method signature, which is the first four bytes of the SHA-512/256 hash of the method signature.

Returns first four bytes of the method signature hash

Return type bytes

get_txn_calls() → int

Returns the number of transactions needed to invoke this ABI method.

static from_json(*resp: Union[str, bytes, bytearray]*) → *algosdk.abi.method.Method*

static from_signature(*s: str*) → *algosdk.abi.method.Method*

dictify() → dict

static undictify(*d: dict*) → *algosdk.abi.method.Method*

class Argument(*arg_type: str, name: Optional[str] = None, desc: Optional[str] = None*)

Bases: object

Represents an argument for a ABI method

Parameters

- **arg_type** (*string*) – ABI type or transaction string of the method argument
- **name** (*string, optional*) – name of this method argument
- **desc** (*string, optional*) – description of this method argument

dictify() → dict

static undictify(*d: dict*) → *algosdk.abi.method.Argument*

class Returns(*arg_type: str, desc: Optional[str] = None*)

Bases: object

Represents a return type for a ABI method

Parameters

- **arg_type** (*string*) – ABI type of this return argument
- **desc** (*string, optional*) – description of this return argument

```

VOID = 'void'
dictify() → dict
static undictify(d: dict) → algosdk.abi.method.Returns

```

8.1.1.10 reference

```

class ABIReferenceType
    Bases: object
    ACCOUNT = 'account'
    APPLICATION = 'application'
    ASSET = 'asset'
is_abi_reference_type(t: Any) → bool

```

8.1.1.11 string_type

```

class StringType
    Bases: algosdk.abi.base_type.ABType
    Represents a String ABI Type for encoding.
    byte_len() → NoReturn
        Return the length in bytes of the ABI type.
    is_dynamic() → bool
        Return whether the ABI type is dynamic.
    encode(string_val: str) → bytes
        Encode a value into a String ABI bytestring.
        Parameters value (str) – string to be encoded.
        Returns encoded bytes of the string
        Return type bytes
    decode(bytestring: Union[bytes, bytearray]) → str
        Decodes a bytestring to a string.
        Parameters bytestring (bytes | bytearray) – bytestring to be decoded
        Returns string from the encoded bytestring
        Return type str

```

8.1.1.12 transaction

```

class ABITransactionType
    Bases: object
    ANY = 'txn'
    PAY = 'pay'
    KEYREG = 'keyreg'
    ACFG = 'acfg'

```

AXFER = 'axfer'

AFRZ = 'afrz'

APPL = 'appl'

is_abi_transaction_type(*t: Any*) → bool

check_abi_transaction_type(*t: Any*, *txn: algosdk.future.transaction.Transaction*) → bool

8.1.1.13 tuple_type

class TupleType(*arg_types: List[Any]*)

Bases: *algosdk.abi.base_type.ABIStructType*

Represents a Tuple ABI Type for encoding.

Parameters **arg_types** (*list*) – list of types in the tuple.

child_types

Type list

byte_len() → int

Return the length in bytes of the ABI type.

is_dynamic() → bool

Return whether the ABI type is dynamic.

encode(*values: Union[List[Any], bytes, bytearray]*) → bytes

Encodes a list of values into a TupleType ABI bytestring.

Parameters

- **values** (*list | bytes | bytearray*) – list of values to be encoded.
- **length of the list cannot exceed a uint16.** (*The*) –
- **the child types are ByteType** (*If*) –
- **bytes or bytearray can be** (*then*) –
- **in to be encoded as well.** (*passed*) –

Returns encoded bytes of the tuple

Return type bytes

decode(*bytestring: Union[bytes, bytearray]*) → list

Decodes a bytestring to a tuple list.

Parameters **bytestring** (*bytes | bytearray*) – bytestring to be decoded

Returns values from the encoded bytestring

Return type list

8.1.1.14 ufixed_type

class UfixedType(*type_size: int, type_precision: int*)

Bases: [algosdk.abi.base_type.ABType](#)

Represents an Ufixed ABI Type for encoding.

Parameters

- **type_size** (*int*) – size of a ufixed type.
- **type_precision** (*int*) – number of precision for a ufixed type.

bit_size

Type *int*

precision

Type *int*

byte_len() → *int*

Return the length in bytes of the ABI type.

is_dynamic() → *bool*

Return whether the ABI type is dynamic.

encode(*value: int*) → *bytes*

Encodes a value into a Ufixed ABI type bytestring. The precision denotes the denominator and the value denotes the numerator.

Parameters **value** (*int*) – ufixed numerator value in uint to be encoded

Returns encoded bytes of the ufixed numerator

Return type *bytes*

decode(*bytestring: Union[bytes, bytearray]*) → *int*

Decodes a bytestring to a ufixed numerator.

Parameters **bytestring** (*bytes | bytearray*) – bytestring to be decoded

Returns ufixed numerator value from the encoded bytestring

Return type *int*

8.1.1.15 uint_type

class UIntType(*type_size: int*)

Bases: [algosdk.abi.base_type.ABType](#)

Represents an UInt ABI Type for encoding.

Parameters **bit_size** (*int*) – size of a uint type, e.g. for a uint8, the bit_size is 8.

bit_size

Type *int*

byte_len() → *int*

Return the length in bytes of the ABI type.

is_dynamic() → *bool*

Return whether the ABI type is dynamic.

encode(*value: int*) → bytes

Encodes a value into a Uint ABI type bytestring.

Parameters **value** (*int*) – uint value to be encoded

Returns encoded bytes of the uint value

Return type bytes

decode(*bytestring: Union[bytes, bytearray]*) → int

Decodes a bytestring to a uint.

Parameters **bytestring** (*bytes | bytearray*) – bytestring to be decoded

Returns uint value from the encoded bytestring

Return type int

8.1.2 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

8.1.3 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=None, round_num=None, **kwargs*)

Return node status immediately after blockNum.

Parameters

- **block_num** (*int, optional*) – block number
- **round_num** (*int, optional*) – alias for block_num; specify one of these

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=None*, *round_num=None*, ***kwargs*)

Return block information.

Parameters

- **round** (*int*, *optional*) – block number; deprecated, please use round_num
- **round_num** (*int*, *optional*) – alias for round; specify only one of these

block_raw(*round=None*, *round_num=None*, ***kwargs*)

Return decoded raw block as the network sees it.

Parameters

- **round** (*int*, *optional*) – block number; deprecated, please use round_num
- **round_num** (*int*, *optional*) – alias for round; specify only one of these

8.1.4 atomic_transaction_composer

class AtomicTransactionComposerStatus(*value*)

Bases: *enum.IntEnum*

An enumeration.

BUILDING = 0

BUILT = 1

SIGNED = 2

SUBMITTED = 3

COMMITTED = 4

populate_foreign_array(*value_to_add: algosdk.atomic_transaction_composer.T*, *foreign_array:*

List[algosdk.atomic_transaction_composer.T], *zero_value:*

Optional[algosdk.atomic_transaction_composer.T] = None) → int

Add a value to an application call's foreign array. The addition will be as compact as possible, and this function will return an index used to reference *value_to_add* in the *foreign_array*.

Parameters

- **value_to_add** – value to add to the array. If the value is already present, it will not be added again. Instead, the existing index will be returned.
- **foreign_array** – the existing foreign array. This input may be modified to append *value_to_add*.
- **zero_value** – If provided, this value indicates two things: the 0 value is reserved for this array so *foreign_array* must start at index 1; additionally, if *value_to_add* equals *zero_value*, then *value_to_add* will not be added to the array and the 0 index will be returned.

class AtomicTransactionComposer

Bases: object

Constructs an atomic transaction group which may contain a combination of Transactions and ABI Method calls.

Parameters

- **status** ([AtomicTransactionComposerStatus](#)) – IntEnum representing the current state of the composer
- **method_dict** (*dict*) – dictionary of an index in the transaction list to a Method object
- **txn_list** (*list* [[TransactionWithSigner](#)]) – list of transactions with signers
- **signed_txns** (*list* [[SignedTransaction](#)]) – list of signed transactions
- **tx_ids** (*list* [*str*]) – list of individual transaction IDs in this atomic group

MAX_GROUP_SIZE = 16

MAX_APP_ARG_LIMIT = 16

get_status() → [algosdk.atomic_transaction_composer.AtomicTransactionComposerStatus](#)

Returns the status of this composer's transaction group.

get_tx_count() → int

Returns the number of transactions currently in this atomic group.

clone() → [algosdk.atomic_transaction_composer.AtomicTransactionComposer](#)

Creates a new composer with the same underlying transactions. The new composer's status will be BUILDING, so additional transactions may be added to it.

add_transaction(*txn_and_signer*: [algosdk.atomic_transaction_composer.TransactionWithSigner](#)) → [algosdk.atomic_transaction_composer.AtomicTransactionComposer](#)

Adds a transaction to this atomic group.

An error will be thrown if the composer's status is not BUILDING, or if adding this transaction causes the current group to exceed MAX_GROUP_SIZE.

Parameters **txn_and_signer** ([TransactionWithSigner](#)) –

```
add_method_call(app_id: int, method: algosdk.abi.method.Method, sender: str, sp:
    algosdk.future.transaction.SuggestedParams, signer:
    algosdk.atomic_transaction_composer.TransactionSigner, method_args:
    Optional[List[Union[Any,
    algosdk.atomic_transaction_composer.TransactionWithSigner]]] = None, on_complete:
    algosdk.future.transaction.OnComplete = <OnComplete.NoOpOC: 0>, local_schema:
    Optional[algosdk.future.transaction.StateSchema] = None, global_schema:
    Optional[algosdk.future.transaction.StateSchema] = None, approval_program:
    Optional[bytes] = None, clear_program: Optional[bytes] = None, extra_pages:
    Optional[int] = None, accounts: Optional[List[str]] = None, foreign_apps:
    Optional[List[int]] = None, foreign_assets: Optional[List[int]] = None, note:
    Optional[bytes] = None, lease: Optional[bytes] = None, rekey_to: Optional[str] = None)
    → algosdk.atomic\_transaction\_composer.AtomicTransactionComposer
```

Add a smart contract method call to this atomic group.

An error will be thrown if the composer's status is not BUILDING, if adding this transaction causes the current group to exceed MAX_GROUP_SIZE, or if the provided arguments are invalid for the given method.

Parameters

- **app_id** (*int*) – application id of app that the method is being invoked on
- **method** (*Method*) – ABI method object with initialized arguments and return types
- **sender** (*str*) – address of the sender
- **sp** (*SuggestedParams*) – suggested params from algod
- **signer** (*TransactionSigner*) – signer that will sign the transactions
- **method_args** (*list[ABIValue | TransactionWithSigner], optional*) – list of arguments to be encoded or transactions that immediately precede this method call
- **on_complete** (*OnComplete, optional*) – intEnum representing what app should do on completion and if blank, it will default to a NoOp call
- **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
- **extra_pages** (*int, optional*) – additional program space for supporting larger programs. A page is 1024 bytes.
- **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
- **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

build_group() → list

Finalize the transaction group and returns the finalized transactions with signers. The composer's status will be at least BUILT after executing this method.

Returns list of transactions with signers

Return type list[*TransactionWithSigner*]

gather_signatures() → list

Obtain signatures for each transaction in this group. If signatures have already been obtained, this method will return cached versions of the signatures. The composer's status will be at least SIGNED after executing this method. An error will be thrown if signing any of the transactions fails.

Returns list of signed transactions

Return type list[SignedTransactions]

submit(*client*: *algosdk.v2client.algod.AlgodClient*) → list

Send the transaction group to the network, but don't wait for it to be committed to a block. An error will be thrown if submission fails. The composer's status must be SUBMITTED or lower before calling this method. If submission is successful, this composer's status will update to SUBMITTED.

Note: a group can only be submitted again if it fails.

Parameters *client* (*AlgodClient*) – Algod V2 client

Returns list of submitted transactions

Return type list[*Transaction*]

execute(*client*: *algosdk.v2client.algod.AlgodClient*, *wait_rounds*: *int*) → *algosdk.atomic_transaction_composer.AtomicTransactionResponse*

Send the transaction group to the network and wait until it's committed to a block. An error will be thrown if submission or execution fails. The composer's status must be SUBMITTED or lower before calling this method, since execution is only allowed once. If submission is successful, this composer's status will update to SUBMITTED. If the execution is also successful, this composer's status will update to COMMITTED.

Note: a group can only be submitted again if it fails.

Parameters

- *client* (*AlgodClient*) – Algod V2 client
- *wait_rounds* (*int*) – maximum number of rounds to wait for transaction confirmation

Returns

Object with confirmed round for this transaction, a list of txIDs of the submitted transactions, and an array of results for each method call transaction in this group. If a method has no return value (void), then the method results array will contain None for that method's return value.

Return type *AtomicTransactionResponse*

class TransactionSigner

Bases: *abc.ABC*

Represents an object which can sign transactions from an atomic transaction group.

abstract sign_transactions(*txn_group*: List[*algosdk.future.transaction.Transaction*], *indexes*: List[int]) → list

class AccountTransactionSigner(*private_key*: str)

Bases: *algosdk.atomic_transaction_composer.TransactionSigner*

Represents a Transaction Signer for an account that can sign transactions from an atomic transaction group.

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

sign_transactions (`txn_group`: `List[algosdk.future.transaction.Transaction]`, `indexes`: `List[int]`) → list
Sign transactions in a transaction group given the indexes.

Returns an array of encoded signed transactions. The length of the array will be the same as the length of `indexesToSign`, and each index `i` in the array corresponds to the signed transaction from `txn_Group[indexesToSign[i]]`.

Parameters

- **txn_group** (`list[Transaction]`) – atomic group of transactions
- **indexes** (`list[int]`) – array of indexes in the atomic transaction group that should be signed

class LogicSigTransactionSigner (`lsig`: `algosdk.future.transaction.LogicSigAccount`)

Bases: `algosdk.atomic_transaction_composer.TransactionSigner`

Represents a Transaction Signer for a LogicSig that can sign transactions from an atomic transaction group.

Parameters `lsig` (`LogicSigAccount`) – LogicSig account

sign_transactions (`txn_group`: `List[algosdk.future.transaction.Transaction]`, `indexes`: `List[int]`) → list
Sign transactions in a transaction group given the indexes.

Returns an array of encoded signed transactions. The length of the array will be the same as the length of `indexesToSign`, and each index `i` in the array corresponds to the signed transaction from `txn_Group[indexesToSign[i]]`.

Parameters

- **txn_group** (`list[Transaction]`) – atomic group of transactions
- **indexes** (`list[int]`) – array of indexes in the atomic transaction group that should be signed

class MultisigTransactionSigner (`msig`: `algosdk.future.transaction.Multisig`, `sks`: `str`)

Bases: `algosdk.atomic_transaction_composer.TransactionSigner`

Represents a Transaction Signer for a Multisig that can sign transactions from an atomic transaction group.

Parameters

- **msig** (`Multisig`) – Multisig account
- **sks** (`str`) – private keys of multisig

sign_transactions (`txn_group`: `List[algosdk.future.transaction.Transaction]`, `indexes`: `List[int]`) → list
Sign transactions in a transaction group given the indexes.

Returns an array of encoded signed transactions. The length of the array will be the same as the length of `indexesToSign`, and each index `i` in the array corresponds to the signed transaction from `txn_Group[indexesToSign[i]]`.

Parameters

- **txn_group** (`list[Transaction]`) – atomic group of transactions
- **indexes** (`list[int]`) – array of indexes in the atomic transaction group that should be signed

```
class TransactionWithSigner(txn: algosdk.future.transaction.Transaction, signer:
                           algosdk.atomic_transaction_composer.TransactionSigner)
    Bases: object

class ABIResult(tx_id: int, raw_value: bytes, return_value: Any, decode_error: Optional[Exception], tx_info:
                dict)
    Bases: object

class AtomicTransactionResponse(confirmed_round: int, tx_ids: List[str], results:
                                List[algosdk.atomic_transaction_composer.ABIResult])
    Bases: object
```

8.1.5 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*)

8.1.6 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', '/genesis']

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

APPID_PREFIX = b'appID'

application ID prefix when signing

Type bytes

HASH_LEN = 32

how long various hash-like fields should be

Type int

CHECK_SUM_LEN_BYTES = 4

how long checksums should be

Type int

KEN_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

8.1.7 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

8.1.8 error

exception **BadTxnSenderError**

Bases: Exception

exception **InvalidThresholdError**

Bases: Exception

exception **InvalidSecretKeyError**

Bases: Exception

exception **MergeKeysMismatchError**

Bases: Exception

exception MergeAuthAddrMismatchError

Bases: Exception

exception DuplicateSigMismatchError

Bases: Exception

exception LogicSigOverspecifiedSignature

Bases: Exception

exception LogicSigSigningKeyMissing

Bases: Exception

exception WrongAmountType

Bases: Exception

exception WrongChecksumError

Bases: Exception

exception WrongKeyLengthError

Bases: Exception

exception WrongMnemonicLengthError

Bases: Exception

exception WrongHashLengthError

Bases: Exception

General error that is normally changed to be more specific

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 OverspecifiedRoundError(contract_type)
    Bases: Exception

exception UnderspecifiedRoundError(contract_type)
    Bases: Exception

exception ZeroAddressError
    Bases: Exception

exception KeyregOnlineTxnInitError(attr)
    Bases: Exception

exception TemplateInputError
    Bases: Exception

exception TemplateError
    Bases: Exception

exception KMDHTTPError
    Bases: Exception

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

exception AlgodResponseError(msg)
    Bases: Exception

exception IndexerHTTPError
    Bases: Exception

exception ConfirmationTimeoutError
    Bases: Exception

exception ABITypeError(msg)
    Bases: Exception

exception ABIEncodingError(msg)
    Bases: Exception

exception AtomicTransactionComposerError(msg)
    Bases: Exception

```

8.1.9 future

8.1.9.1 future.template

```

class Template
    Bases: object

    NOTE: This class is deprecated

    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

    NOTE: This class is deprecated.

```

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

$$\text{first_recipient_amount} * \text{rat_2} == \text{second_recipient_amount} * \text{rat_1}$$

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](#)

NOTE: This class is deprecated.

Hash Time Locked Contract allows a user to receive 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 transferred 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: Optional[str] = None*)

Bases: *algosdk.future.template.Template*

NOTE: This class is deprecated.

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 receives 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*

NOTE: This class is deprecated.

PeriodicPayment contract enables creating an account which allows the withdrawal of a fixed amount of assets every fixed number of rounds to a specific Algorand 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](#)

NOTE: This class is deprecated.

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 transferred
- **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*)

8.1.9.2 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 if flat_fee is false). When flat_fee is true, fee may fall to zero but a group of N atomic transactions must still have a fee of at least N*min_txn_fee.
- **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.

static as_hash(*hash*)

Confirm that a value is 32 bytes. If all zeros, or a falsy value, return None

static as_note(*note*)

classmethod as_lease(*lease*)

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*)

static required(*arg*)

static creatable_index(*index, required=False*)

Coerce an index for apps or assets to an integer.

By using this in all constructors, we allow callers to use strings as indexes, check our convenience Txn types to ensure index is set, and ensure that 0 is always used internally for an unset id, not None, so `__eq__` works properly.

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,  
               nonpart=None, sprfkey=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
- **nonpart** (*bool, optional*) – mark the account non-participating if true
- **StateProofPK** – state proof

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`

vote_pk

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
    nonpart
        Type bool
    sprfkey
        Type str
    dictify()
class KeyregOnlineTxn(sender, sp, votekey, selkey, votefst, votelst, votekd, note=None, lease=None,
                      rekey_to=None, sprfkey=None)
Bases: algosdk.future.transaction.KeyregTxn

```

Represents an online key registration transaction. nonpart is implicitly False for this 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
- **sprfkey** (*str*, *optional*) – state proof ID

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

sprfkey
Type str

class KeyregOfflineTxn(*sender, sp, note=None, lease=None, rekey_to=None*)

Bases: [algosdk.future.transaction.KeyregTxn](#)

Represents an offline key registration transaction. nonpart is implicitly False for this transaction.

Parameters

- **sender** (*str*) – address of sender
- **sp** ([SuggestedParams](#)) – suggested params from algod
- **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*

type

Type *str*

lease

Type *byte[32]*

rekey_to

Type *str*

class KeyregNonparticipatingTxn(*sender*, *sp*, *note=None*, *lease=None*, *rekey_to=None*)

Bases: [algosdk.future.transaction.KeyregTxn](#)

Represents a nonparticipating key registration transaction. nonpart is implicitly True for this transaction.

Parameters

- **sender** (*str*) – address of sender
- **sp** ([SuggestedParams](#)) – suggested params from algod
- **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*

type

Type *str*

lease

Type *byte[32]*

rekey_to

Type *str*

```
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 meta-data (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()
classmethod as_metadata(md)
class AssetCreateTxn(sender, sp, total, decimals, default_frozen, *, manager=None, reserve=None,
                    freeze=None, clawback=None, unit_name="", asset_name="", url="",
                    metadata_hash=None, note=None, lease=None, rekey_to=None)
Bases: algosdk.future.transaction.AssetConfigTxn
Represents a transaction for asset creation.
```

Keyword arguments are required, starting with the special addresses, to prevent errors, as type checks can't prevent simple confusion of similar typed arguments. Since the special addresses are required, `strict_empty_address_check` is turned off.

Parameters

- **sender** (*str*) – address of the sender
- **sp** ([SuggestedParams](#)) – suggested params from algod
- **total** (*int*) – total number of base units of this asset created
- **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.
- **default_frozen** (*bool*) – whether slots for this asset in user accounts are frozen by default
- **manager** (*str*) – address allowed to change nonzero addresses for this asset
- **reserve** (*str*) – account whose holdings of this asset should be reported as “not minted”
- **freeze** (*str*) – account allowed to change frozen state of holdings of this asset
- **clawback** (*str*) – account allowed take units of this asset from any account
- **unit_name** (*str*) – hint for the name of a unit of this asset
- **asset_name** (*str*) – hint for the name of the asset
- **url** (*str*) – a URL where more information about the asset can be retrieved
- **metadata_hash** (*byte[32]*, *optional*) – a commitment to some unspecified asset meta-data (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
- **rekey_to** (*str*, *optional*) – additionally rekey the sender to this address

class `AssetDestroyTxn`(*sender*, *sp*, *index*, *note=None*, *lease=None*, *rekey_to=None*)

Bases: [algosdk.future.transaction.AssetConfigTxn](#)

Represents a transaction for asset destruction.

An asset destruction transaction can only be sent by the manager address, and only when the manager possesses all units of the asset.

class `AssetUpdateTxn`(*sender*, *sp*, *index*, *, *manager*, *reserve*, *freeze*, *clawback*, *note=None*, *lease=None*, *rekey_to=None*)

Bases: [algosdk.future.transaction.AssetConfigTxn](#)

Represents a transaction for asset modification.

To update asset configuration, include the following: `index`, `manager`, `reserve`, `freeze`, `clawback`.

Keyword arguments are required, starting with the special addresses, to prevent argument reordering errors. Since the special addresses are required, `strict_empty_address_check` is turned off.

Parameters

- **sender** (*str*) – address of the sender
- **sp** ([SuggestedParams](#)) – suggested params from algod
- **index** (*int*) – index of the asset to reconfigure

- **manager** (*str*) – address allowed to change nonzero addresses for this asset
- **reserve** (*str*) – account whose holdings of this asset should be reported as “not minted”
- **freeze** (*str*) – account allowed to change frozen state of holdings of this asset
- **clawback** (*str*) – account allowed take units of this asset from any account
- **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

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 (or use
    AssetOptInTxn)
    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 pay-
            ing 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 **AssetOptInTxn**(*sender, sp, index, note=None, lease=None, rekey_to=None*)

Bases: [*algosdk.future.transaction.AssetTransferTxn*](#)

Make a transaction that will opt in to an ASA

Parameters

- **sender** (*str*) – address of sender
- **sp** ([*SuggestedParams*](#)) – contains information such as fee and genesis hash
- **index** (*int*) – the ASA to opt into
- **note** (*bytes, optional*) – transaction note field
- **lease** (*bytes, optional*) – transaction lease field
- **rekey_to** (*str, optional*) – rekey-to field, see Transaction

See `AssetTransferTxn`

class `AssetCloseOutTxn`(*sender, sp, receiver, index, note=None, lease=None, rekey_to=None*)

Bases: `algosdk.future.transaction.AssetTransferTxn`

Make a transaction that will send all of an ASA away, and opt out of it.

Parameters

- **sender** (*str*) – address of sender
- **sp** (`SuggestedParams`) – contains information such as fee and genesis hash
- **receiver** (*str*) – address of the receiver
- **index** (*int*) – the ASA to opt into
- **note** (*bytes, optional*) – transaction note field
- **lease** (*bytes, optional*) – transaction lease field
- **rekey_to** (*str, optional*) – rekey-to field, see `Transaction`

See `AssetTransferTxn`

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`(*value*)

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, extra_pages=0)
```

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
- **extra_pages** (*int*, *optional*) – additional program space for supporting larger programs. A page is 1024 bytes.

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]
    extra_pages
        Type int
    static state_schema(schema)
        Confirm the argument is a StateSchema, or false which is coerced to None
    static teal_bytes(teal)
        Confirm the argument is bytes-like, or false which is coerced to None
    static bytes_list(lst)
        Confirm or coerce list elements to bytes. Return None for empty/false lst.
    static int_list(lst)
        Confirm or coerce list elements to int. Return None for empty/false lst.
    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, extra_pages=0)
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 do 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*) – restricts 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
- **extra_pages** (*int*, *optional*) – provides extra program size

See `ApplicationCallTxn`

```
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`

```
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`

```
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`

```
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`

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 for 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`

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

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: [algosdk.future.transaction.Transaction](#), multisig: [algosdk.future.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](#)

auth_addr

Type *str, optional*

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*: *List*[algosdk.future.transaction.MultisigTransaction]) → *algosdk.future.transaction.MultisigTransaction*

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

    NOTE: This type is deprecated. Use LogicSigAccount instead.

    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
- **LogicSigOverspecifiedSignature** – if the opposite signature type has already been provided

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 LogicSigAccount(*program*: bytes, *args*: Optional[List[bytes]] = None)

Bases: object

Represents an account that can sign with a LogicSig program.

dictify()

static undictify(*d*)

is_delegated() → bool

Check if this LogicSigAccount has been delegated to another account with a signature.

Returns True if and only if this is a delegated LogicSigAccount.

Return type bool

verify() → bool

Verifies the LogicSig's program and signatures.

Returns

True if and only if the LogicSig program and signatures are valid.

Return type bool

address() → str

Get the address of this LogicSigAccount.

If the LogicSig is delegated to another account, this will return the address of that account.

If the LogicSig is not delegated to another account, this will return an escrow address that is the hash of the LogicSig's program code.

sign_multisig(*multisig*: `algosdk.future.transaction.Multisig`, *private_key*: str) → None

Turns this LogicSigAccount into a delegated LogicSig.

This type of LogicSig has the authority to sign transactions on behalf of another account, called the delegating account. Use this function if the delegating account is a multisig account.

Parameters

- **multisig** (`Multisig`) – The multisig delegating account
- **private_key** (str) – The private key of one of the members of the delegating multisig account. Use *append_to_multisig* to add additional signatures from other members.

Raises

- *InvalidSecretKeyError* – if no matching private key in multisig object
- *LogicSigOverspecifiedSignature* – if this LogicSigAccount has already been signed with a single private key.

append_to_multisig(*private_key*: str) → None

Adds an additional signature from a member of the delegating multisig account.

Parameters **private_key** (str) – The private key of one of the members of the delegating multisig account.

Raises *InvalidSecretKeyError* – if no matching private key in multisig object

sign(*private_key*: str) → None

Turns this LogicSigAccount into a delegated LogicSig.

This type of LogicSig has the authority to sign transactions on behalf of another account, called the delegating account. If the delegating account is a multisig account, use *sign_multisig* instead.

Parameters **private_key** (str) – The private key of the delegating account.

Raises *LogicSigOverspecifiedSignature* – if this LogicSigAccount has already been signed by a multisig account.

```
class LogicSigTransaction(transaction: algosdk.future.transaction.Transaction, lsig:
    Union[algosdk.future.transaction.LogicSig,
    algosdk.future.transaction.LogicSigAccount])
```

Bases: object

Represents a logic signed transaction

Parameters

- **transaction** (`Transaction`) –
- **lsig** (`LogicSig` or `LogicSigAccount`) –

transaction

Type *Transaction*

lsig

Type *LogicSig*

auth_addr

Type str, optional

verify() → bool

Verify the LogicSig used to sign the transaction

Returns true if the signature is valid, 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)

wait_for_confirmation(*algod_client*: `algosdk.v2client.algod.AlgodClient`, *txid*: *str*, *wait_rounds*: *int* = 0, ***kwargs*)

Block until a pending transaction is confirmed by the network.

Parameters

- **algod_client** (`algod.AlgodClient`) – Instance of the *algod* client
- **txid** (*str*) – transaction ID
- **wait_rounds** (*int*, *optional*) – The number of rounds to block for before exiting with an Exception. If not supplied, this will be 1000.

create_dryrun(*client*: `algosdk.v2client.algod.AlgodClient`, *txns*: *List[Union[algosdk.future.transaction.SignedTransaction, algosdk.future.transaction.LogicSigTransaction]]*, *protocol_version*=None, *latest_timestamp*=None, *round*=None) → `algosdk.v2client.models.dryrun_request.DryrunRequest`
Create DryrunRequest object from a client and list of signed transactions

Parameters

- **algod_client** (`algod.AlgodClient`) – Instance of the *algod* client
- **txns** (*List[SignedTransaction]*) – transaction ID
- **protocol_version** (*string*, *optional*) – The protocol version to evaluate against
- **latest_timestamp** (*int*, *optional*) – The latest timestamp to evaluate against
- **round** (*int*, *optional*) – The round to evaluate against

decode_programs(*app*)

8.1.10 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*

8.1.11 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)`

`check_push_int_block(program, pc)`

`read_push_int_block(program, pc)`

`check_push_byte_block(program, pc)`

`read_push_byte_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

`get_application_address(appID: int) → str`

Return the escrow address of an application.

Parameters `appID` (int) – The ID of the application.

Returns The address corresponding to that application's escrow account.

Return type str

8.1.12 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. This method returns the Algorand address and will be deprecated, use `account.address_from_private_key` instead.

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

Returns public key in base32

Return type str

8.1.13 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 transferring to two predefined addresses in a specified ratio such that for the given ratn and ratd parameters we have:

$$\text{first_recipient_amount} * \text{rat_2} == \text{second_recipient_amount} * \text{rat_1}$$

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 receive 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 transferred 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: Optional[int] = None, close_remainder_address: Optional[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 receives 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 Algorand 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 transferred
- **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*)

8.1.14 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). When *flat_fee* is true, fee may fall to zero but a group of *N* atomic transactions must still have a fee of at least *N***min_txn_fee*.
- **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). When flat_fee is true, fee may fall to zero but a group of N atomic transactions must still have a fee of at least N*min_txn_fee.
    • 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). When flat_fee is true, fee may fall to zero but a group of N atomic transactions must still have a fee of at least $N * \text{min_txn_fee}$.
- **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 meta-data (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). When flat_fee is true, fee may fall to zero but a group of N atomic transactions must still have a fee of at least N*min_txn_fee.
- **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). When `flat_fee` is true, fee may fall to zero but a group of `N` atomic transactions must still have a fee of at least `N*min_txn_fee`.
- **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)

8.1.15 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

build_headers_from(*kwarg_headers: Dict[str, Any], additional_headers: Dict[str, Any]*)

Build correct headers for *AlgodClient.algod_request*.

Parameters

- **kwarg_headers** (*Dict[str, Any]*) – headers passed through kwargs.
- **additional_headers** (*Dict[str, Any]*) – additional headers to pass to *AlgodClient.algod_request*

Returns final version of headers dictionary to be used for *AlgodClient.algod_request*

Return type Dict[str, any]

8.1.16 v2client

8.1.16.1 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

asset_info(*asset_id*, ***kwargs*)

Return information about a specific asset.

Parameters **asset_id** (*int*) – The ID of the asset to look up.

application_info(*application_id*, ***kwargs*)

Return information about a specific application.

Parameters **application_id** (*int*) – The ID of the application to look up.

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=None*, *response_format='json'*, *round_num=None*, ***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”
- **round_num** (*int*, *optional*) – alias for block; specify one of these

ledger_supply(***kwargs*)

Return supply details for node’s ledger.

status(***kwargs*)

Return node status.

status_after_block(*block_num=None*, *round_num=None*, ***kwargs*)

Return node status immediately after blockNum.

Parameters

- **block_num** – block number
- **round_num** (*int*, *optional*) – alias for block_num; specify one of these

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 loaded from json response body. “result” property contains compiled bytes, “hash” - program hash (escrow address)

Return type dict

dryrun(*drr*, ***kwargs*)

Dryrun with remote algod.

Parameters

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

Returns loaded from json response body

Return type dict

genesis(***kwargs*)

Returns the entire genesis file.

proof(*round_num*, *txid*, ***kwargs*)

Get the proof for a given transaction in a round.

Parameters

- **round_num** (*int*) – The round in which the transaction appears.
- **txid** (*str*) – The transaction ID for which to generate a proof.

8.1.16.2 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, round_num=None, include_all=False, **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 (results with an amount equal to this value are excluded)
- **max_balance**(int, optional) – results should have an amount less than this value (results with an amount equal to this value are excluded)
- **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
- **round_num**(int, optional) – alias for block; only specify one of these
- **include_all**(bool, optional) – include all items including closed accounts, deleted applications, destroyed assets, opted-out asset holdings, and closed-out application local-states. Defaults to false.

asset_balances(asset_id, limit=None, next_page=None, min_balance=None, max_balance=None, block=None, round_num=None, include_all=False, **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 (results with an amount equal to this value are excluded)
- **max_balance**(int, optional) – results should have an amount less than this value (results with an amount equal to this value are excluded)
- **block**(int, optional) – include results for the specified round; for performance reasons, this parameter may be disabled on some configurations
- **round_num**(int, optional) – alias for block; only specify one of these

- **include_all** (*bool, optional*) – include all items including closed accounts, deleted applications, destroyed assets, opted-out asset holdings, and closed-out application local-states. Defaults to false.

block_info(*block=None, round_num=None, **kwargs*)

Get the block for the given round.

Parameters

- **block** (*int, optional*) – block number
- **round_num** (*int, optional*) – alias for block; specify one of these

account_info(*address, block=None, round_num=None, include_all=False, **kwargs*)

Return account information.

Parameters

- **address** (*str*) – account public key
- **block** (*int, optional*) – use results from the specified round
- **round_num** (*int, optional*) – alias for block; only specify one of these
- **include_all** (*bool, optional*) – include all items including closed accounts, deleted applications, destroyed assets, opted-out asset holdings, and closed-out application local-states. Defaults to false.

transaction(*txid, **kwargs*)

Returns information about the given transaction.

Parameters **txid** (*str*) – The ID of the transaction to look up.

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, round_num=None, **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*) – include results which include the rekey-to field
- **round_num** (*int, optional*) – alias for block; only specify one of these

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, round_num=None, **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*) – include results which include the rekey-to field
- **round_num** (*int, optional*) – alias for block; only specify one of these

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, round_num=None, **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*) – include results which include the rekey-to field
- **round_num** (*int*, *optional*) – alias for block; only specify one of these

search_assets(*limit=None, next_page=None, creator=None, name=None, unit=None, asset_id=None, include_all=False, **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
- **include_all** (*bool*, *optional*) – include all items including closed accounts, deleted applications, destroyed assets, opted-out asset holdings, and closed-out application local-states. Defaults to false.

asset_info(*asset_id, include_all=False, **kwargs*)

Return asset information.

Parameters

- **asset_id** (*int*) – asset index
- **include_all** (*bool*, *optional*) – include all items including closed accounts, deleted applications, destroyed assets, opted-out asset holdings, and closed-out application local-states. Defaults to false.

applications(*application_id, round=None, round_num=None, include_all=False, **kwargs*)

Return applications that satisfy the conditions.

Parameters

- **application_id** (*int*) – application index
- **round** (*int*, *optional*) – not supported, DO NOT USE!
- **round_num** (*int*, *optional*) – not supported, DO NOT USE!
- **include_all** (*bool*, *optional*) – include all items including closed accounts, deleted applications, destroyed assets, opted-out asset holdings, and closed-out application local-states. Defaults to false.

search_applications(*application_id=None, round=None, limit=None, next_page=None, round_num=None, include_all=False, **kwargs*)

Return applications that satisfy the conditions.

Parameters

- **application_id** (*int, optional*) – restrict search to application index
- **round** (*int, optional*) – not supported, DO NOT USE!
- **limit** (*int, optional*) – restrict number of results to limit
- **next_page** (*string, optional*) – used for pagination
- **round_num** (*int, optional*) – not supported, DO NOT USE!
- **include_all** (*bool, optional*) – include all items including closed accounts, deleted applications, destroyed assets, opted-out asset holdings, and closed-out application local-states. Defaults to false.

application_logs(*application_id, limit=None, min_round=None, max_round=None, next_page=None, sender_addr=None, txid=None, **kwargs*)

Return log messages generated by the passed in application.

Parameters

- **application_id** (*int*) – application index
- **limit** (*int, optional*) – limit maximum number of results to return
- **min_round** (*int, optional*) – only include results at or after the specified round
- **max_round** (*int, optional*) – only include results at or before the specified round
- **next_page** (*string, optional*) – used for pagination
- **sender_addr** (*string, optional*) – only include transactions with this sender address
- **txid** (*string, optional*) – only include results with this transaction ID

8.1.17 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

8.1.18 wordlist

word_list_raw()

Return the wordlist used for mnemonics.

PYTHON MODULE INDEX

a

- `algosdk.abi.address_type`, 17
- `algosdk.abi.array_dynamic_type`, 18
- `algosdk.abi.array_static_type`, 18
- `algosdk.abi.base_type`, 19
- `algosdk.abi.bool_type`, 20
- `algosdk.abi.byte_type`, 20
- `algosdk.abi.contract`, 21
- `algosdk.abi.interface`, 21
- `algosdk.abi.method`, 22
- `algosdk.abi.reference`, 23
- `algosdk.abi.string_type`, 23
- `algosdk.abi.transaction`, 23
- `algosdk.abi.tuple_type`, 24
- `algosdk.abi.ufixed_type`, 25
- `algosdk.abi.uint_type`, 25
- `algosdk.account`, 26
- `algosdk.algod`, 26
- `algosdk.atomic_transaction_composer`, 29
- `algosdk.auction`, 34
- `algosdk.constants`, 35
- `algosdk.encoding`, 38
- `algosdk.error`, 39
- `algosdk.future.template`, 41
- `algosdk.future.transaction`, 45
- `algosdk.kmd`, 71
- `algosdk.logic`, 75
- `algosdk.mnemonic`, 77
- `algosdk.template`, 77
- `algosdk.transaction`, 81
- `algosdk.util`, 94
- `algosdk.v2client.algod`, 95
- `algosdk.v2client.indexer`, 98
- `algosdk.wallet`, 104
- `algosdk.wordlist`, 107

A

ABIEncodingError, 41
 ABIReferenceType (class in *algosdk.abi.reference*), 23
 ABIResult (class in *algosdk.atomic_transaction_composer*), 34
 ABITransactionType (class in *algosdk.abi.transaction*), 23
 ABIType (class in *algosdk.abi.base_type*), 19
 ABITypeError, 41
 ACCOUNT (*ABIReferenceType* attribute), 23
 account_info() (*AlgodClient* method), 28, 95
 account_info() (*IndexerClient* method), 100
 accounts (*ApplicationCallTxn* attribute), 61
 accounts() (*IndexerClient* method), 99
 AccountTransactionSigner (class in *algosdk.atomic_transaction_composer*), 32
 ACFG (*ABITransactionType* attribute), 23
 add_method_call() (*AtomicTransactionComposer* method), 30
 add_transaction() (*AtomicTransactionComposer* method), 30
 address() (in module *algosdk.logic*), 76
 address() (*LogicSig* method), 68, 92
 address() (*LogicSigAccount* method), 68
 address() (*Multisig* method), 66, 91
 address_from_private_key() (in module *algosdk.account*), 26
 ADDRESS_LEN (in module *algosdk.constants*), 37
 AddressType (class in *algosdk.abi.address_type*), 17
 AFRZ (*ABITransactionType* attribute), 24
 algod_address (*AlgodClient* attribute), 26, 95
 ALGOD_AUTH_HEADER (in module *algosdk.constants*), 35
 algod_request() (*AlgodClient* method), 26, 95
 algod_token (*AlgodClient* attribute), 26, 95
 AlgodClient (class in *algosdk.algod*), 26
 AlgodClient (class in *algosdk.v2client.algod*), 95
 AlgodHTTPError, 41
 AlgodResponseError, 41
 algos_to_microalgos() (in module *algosdk.util*), 94
 algosdk.abi.address_type
 module, 17
 algosdk.abi.array_dynamic_type
 module, 18
 algosdk.abi.array_static_type
 module, 18
 algosdk.abi.base_type
 module, 19
 algosdk.abi.bool_type
 module, 20
 algosdk.abi.byte_type
 module, 20
 algosdk.abi.contract
 module, 21
 algosdk.abi.interface
 module, 21
 algosdk.abi.method
 module, 22
 algosdk.abi.reference
 module, 23
 algosdk.abi.string_type
 module, 23
 algosdk.abi.transaction
 module, 23
 algosdk.abi.tuple_type
 module, 24
 algosdk.abi.ufixed_type
 module, 25
 algosdk.abi.uint_type
 module, 25
 algosdk.account
 module, 26
 algosdk.algod
 module, 26
 algosdk.atomic_transaction_composer
 module, 29
 algosdk.auction
 module, 34
 algosdk.constants
 module, 35
 algosdk.encoding
 module, 38
 algosdk.error
 module, 39
 algosdk.future.template

module, 41
 algosdk.future.transaction
 module, 45
 algosdk.kmd
 module, 71
 algosdk.logic
 module, 75
 algosdk.mnemonic
 module, 77
 algosdk.template
 module, 77
 algosdk.transaction
 module, 81
 algosdk.util
 module, 94
 algosdk.v2client.algod
 module, 95
 algosdk.v2client.indexer
 module, 98
 algosdk.wallet
 module, 104
 algosdk.wordlist
 module, 107
 amount (*AssetTransferTxn* attribute), 58, 89
 amt (*PaymentTxn* attribute), 47, 83
 ANY (*ABITransactionType* attribute), 23
 app_args (*ApplicationCallTxn* attribute), 61
 APPCALL_TXN (in module *algosdk.constants*), 36
 append_to_multisig() (*LogicSig* method), 68, 93
 append_to_multisig() (*LogicSigAccount* method), 69
 APPID_PREFIX (in module *algosdk.constants*), 37
 APPL (*ABITransactionType* attribute), 24
 APPLICATION (*ABIReferenceType* attribute), 23
 application_info() (*AlgodClient* method), 96
 application_logs() (*IndexerClient* method), 104
 ApplicationCallTxn (class in *algosdk.future.transaction*), 59
 ApplicationClearStateTxn (class in *algosdk.future.transaction*), 64
 ApplicationCloseOutTxn (class in *algosdk.future.transaction*), 63
 ApplicationCreateTxn (class in *algosdk.future.transaction*), 61
 ApplicationDeleteTxn (class in *algosdk.future.transaction*), 62
 ApplicationNoOpTxn (class in *algosdk.future.transaction*), 64
 ApplicationOptInTxn (class in *algosdk.future.transaction*), 63
 applications() (*IndexerClient* method), 103
 ApplicationUpdateTxn (class in *algosdk.future.transaction*), 62
 approval_program (*ApplicationCallTxn* attribute), 61
 args (*LogicSig* attribute), 67, 92
 Argument (class in *algosdk.abi.method*), 22
 ArrayDynamicType (class in *algosdk.abi.array_dynamic_type*), 18
 ArrayStaticType (class in *algosdk.abi.array_static_type*), 18
 as_hash() (*Transaction* static method), 46
 as_lease() (*Transaction* class method), 46
 as_metadata() (*AssetConfigTxn* class method), 54
 as_note() (*Transaction* static method), 46
 ASSET (*ABIReferenceType* attribute), 23
 asset_balances() (*IndexerClient* method), 99
 asset_info() (*AlgodClient* method), 28, 96
 asset_info() (*IndexerClient* method), 103
 asset_name (*AssetConfigTxn* attribute), 54, 86
 AssetCloseOutTxn (class in *algosdk.future.transaction*), 59
 ASSETCONFIG_TXN (in module *algosdk.constants*), 36
 AssetConfigTxn (class in *algosdk.future.transaction*), 52
 AssetConfigTxn (class in *algosdk.transaction*), 84
 AssetCreateTxn (class in *algosdk.future.transaction*), 54
 AssetDestroyTxn (class in *algosdk.future.transaction*), 55
 ASSETFREEZE_TXN (in module *algosdk.constants*), 36
 AssetFreezeTxn (class in *algosdk.future.transaction*), 56
 AssetFreezeTxn (class in *algosdk.transaction*), 87
 AssetOptInTxn (class in *algosdk.future.transaction*), 58
 ASSETTRANSFER_TXN (in module *algosdk.constants*), 36
 AssetTransferTxn (class in *algosdk.future.transaction*), 57
 AssetTransferTxn (class in *algosdk.transaction*), 88
 AssetUpdateTxn (class in *algosdk.future.transaction*), 55
 assign_group_id() (in module *algosdk.future.transaction*), 70
 assign_group_id() (in module *algosdk.transaction*), 94
 AtomicTransactionComposer (class in *algosdk.atomic_transaction_composer*), 30
 AtomicTransactionComposerError, 41
 AtomicTransactionComposerStatus (class in *algosdk.atomic_transaction_composer*), 29
 AtomicTransactionResponse (class in *algosdk.atomic_transaction_composer*), 34
 auction_id (*Bid* attribute), 34
 auction_key (*Bid* attribute), 34
 auth_addr (*LogicSigTransaction* attribute), 69
 auth_addr (*MultisigTransaction* attribute), 65
 authorizing_address (*SignedTransaction* attribute), 65, 90
 automate_handle() (*Wallet* method), 107
 AXFER (*ABITransactionType* attribute), 23

B

BadTxnSenderError, 39
 Bid (class in *algosdk.auction*), 34
 bid (SignedBid attribute), 35
 bid_currency (Bid attribute), 34
 bid_id (Bid attribute), 34
 BID_PREFIX (in module *algosdk.constants*), 37
 bidder (Bid attribute), 34
 bit_size (UfixedType attribute), 25
 bit_size (UIntType attribute), 25
 block_info() (AlgodClient method), 29, 96
 block_info() (IndexerClient method), 100
 block_raw() (AlgodClient method), 29
 BoolType (class in *algosdk.abi.bool_type*), 20
 build_group() (AtomicTransactionComposer method), 31
 build_headers_from() (in module *algosdk.util*), 95
 BUILDING (AtomicTransactionComposerStatus attribute), 29
 BUILT (AtomicTransactionComposerStatus attribute), 29
 byte_len() (ABIType method), 19
 byte_len() (AddressType method), 17
 byte_len() (ArrayDynamicType method), 18
 byte_len() (ArrayStaticType method), 18
 byte_len() (BoolType method), 20
 byte_len() (ByteType method), 20
 byte_len() (StringType method), 23
 byte_len() (TupleType method), 24
 byte_len() (UfixedType method), 25
 byte_len() (UIntType method), 25
 bytes_list() (ApplicationCallTxn static method), 61
 BYTES_PREFIX (in module *algosdk.constants*), 37
 ByteType (class in *algosdk.abi.byte_type*), 20

C

calculate_group_id() (in module *algosdk.future.transaction*), 70
 calculate_group_id() (in module *algosdk.transaction*), 94
 check_abi_transaction_type() (in module *algosdk.abi.transaction*), 24
 check_byte_const_block() (in module *algosdk.logic*), 76
 check_int_const_block() (in module *algosdk.logic*), 76
 check_program() (in module *algosdk.logic*), 75
 check_push_byte_block() (in module *algosdk.logic*), 76
 check_push_int_block() (in module *algosdk.logic*), 76
 CHECK_SUM_LEN_BYTES (in module *algosdk.constants*), 37
 checksum() (in module *algosdk.encoding*), 39
 child_type (ArrayDynamicType attribute), 18

child_type (ArrayStaticType attribute), 18
 child_types (TupleType attribute), 24
 clawback (AssetConfigTxn attribute), 54, 86
 clear_program (ApplicationCallTxn attribute), 61
 ClearStateOC (OnComplete attribute), 59
 clone() (AtomicTransactionComposer method), 30
 close_assets_to (AssetTransferTxn attribute), 58, 89
 close_remainder_to (PaymentTxn attribute), 47, 83
 CloseOutOC (OnComplete attribute), 59
 COMMITTED (AtomicTransactionComposerStatus attribute), 29
 compile() (AlgodClient method), 97
 ConfirmationTimeoutError, 41
 consensus_version (SuggestedParams attribute), 45
 Contract (class in *algosdk.abi.contract*), 21
 creatable_index() (Transaction static method), 46
 create_dryrun() (in module *algosdk.future.transaction*), 71
 create_wallet() (KMDClient method), 72

D

decimals (AssetConfigTxn attribute), 54, 87
 decode() (ABIType method), 19
 decode() (AddressType method), 17
 decode() (ArrayDynamicType method), 18
 decode() (ArrayStaticType method), 19
 decode() (BoolType method), 20
 decode() (ByteType method), 20
 decode() (StringType method), 23
 decode() (TupleType method), 24
 decode() (UfixedType method), 25
 decode() (UIntType method), 26
 decode_address() (in module *algosdk.encoding*), 39
 decode_programs() (in module *algosdk.future.transaction*), 71
 default_frozen (AssetConfigTxn attribute), 53, 86
 delete_key() (KMDClient method), 74
 delete_key() (Wallet method), 106
 delete_multisig() (KMDClient method), 75
 delete_multisig() (Wallet method), 106
 DeleteApplicationOC (OnComplete attribute), 59
 dictify() (ApplicationCallTxn method), 61
 dictify() (Argument method), 22
 dictify() (AssetConfigTxn method), 54, 87
 dictify() (AssetFreezeTxn method), 57, 88
 dictify() (AssetTransferTxn method), 58, 90
 dictify() (Bid method), 34
 dictify() (Contract method), 21
 dictify() (Interface method), 21
 dictify() (KeyregTxn method), 49, 84
 dictify() (LogicSig method), 67, 92
 dictify() (LogicSigAccount method), 68
 dictify() (LogicSigTransaction method), 70, 93
 dictify() (Method method), 22

dictify() (*Multisig method*), 67, 91
dictify() (*MultisigSubsig method*), 67, 92
dictify() (*MultisigTransaction method*), 66, 90
dictify() (*NetworkInfo method*), 21
dictify() (*NoteField method*), 35
dictify() (*PaymentTxn method*), 47, 83
dictify() (*Returns method*), 23
dictify() (*SignedBid method*), 35
dictify() (*SignedTransaction method*), 65, 90
dictify() (*StateSchema method*), 59
dictify() (*Transaction method*), 46, 81
dictify() (*TxGroup method*), 70, 94
dryrun() (*AlgodClient method*), 98
DuplicateSigMismatchError, 40
DynamicFee (*class in algosdk.future.template*), 43
DynamicFee (*class in algosdk.template*), 79

E

EmptyAddressError, 40
encode() (*ABType method*), 19
encode() (*AddressType method*), 17
encode() (*ArrayDynamicType method*), 18
encode() (*ArrayStaticType method*), 19
encode() (*BoolType method*), 20
encode() (*ByteType method*), 20
encode() (*StringType method*), 23
encode() (*TupleType method*), 24
encode() (*UfixedType method*), 25
encode() (*UintType method*), 25
encode_address() (*in module algosdk.encoding*), 39
estimate_size() (*Transaction method*), 46, 81
execute() (*AtomicTransactionComposer method*), 32
export_key() (*KMDClient method*), 73
export_key() (*Wallet method*), 105
export_master_derivation_key() (*KMDClient method*), 73
export_master_derivation_key() (*Wallet method*), 105
export_multisig() (*KMDClient method*), 75
export_multisig() (*Wallet method*), 106
extra_pages (*ApplicationCallTxn attribute*), 61

F

fee (*ApplicationCallTxn attribute*), 60
fee (*AssetConfigTxn attribute*), 53, 86
fee (*AssetFreezeTxn attribute*), 56, 87
fee (*AssetTransferTxn attribute*), 57, 89
fee (*KeyregNonparticipatingTxn attribute*), 52
fee (*KeyregOfflineTxn attribute*), 51
fee (*KeyregOnlineTxn attribute*), 49
fee (*KeyregTxn attribute*), 48, 83
fee (*PaymentTxn attribute*), 47, 82
fee (*SuggestedParams attribute*), 45
first (*SuggestedParams attribute*), 45

first_valid_round (*ApplicationCallTxn attribute*), 60
first_valid_round (*AssetConfigTxn attribute*), 53, 86
first_valid_round (*AssetFreezeTxn attribute*), 56, 87
first_valid_round (*AssetTransferTxn attribute*), 57, 89
first_valid_round (*KeyregNonparticipatingTxn attribute*), 52
first_valid_round (*KeyregOfflineTxn attribute*), 51
first_valid_round (*KeyregOnlineTxn attribute*), 50
first_valid_round (*KeyregTxn attribute*), 48, 84
first_valid_round (*PaymentTxn attribute*), 47, 82
flat_fee (*SuggestedParams attribute*), 45
foreign_apps (*ApplicationCallTxn attribute*), 61
foreign_assets (*ApplicationCallTxn attribute*), 61
freeze (*AssetConfigTxn attribute*), 54, 86
from_json() (*Contract static method*), 21
from_json() (*Interface static method*), 21
from_json() (*Method static method*), 22
from_master_derivation_key() (*in module algosdk.mnemonic*), 77
from_private_key() (*in module algosdk.mnemonic*), 77
from_signature() (*Method static method*), 22
from_string() (*ABType static method*), 19
future_msgpack_decode() (*in module algosdk.encoding*), 38

G

gather_signatures() (*AtomicTransactionComposer method*), 32
gen (*SuggestedParams attribute*), 45
generate_account() (*in module algosdk.account*), 26
generate_key() (*KMDClient method*), 73
generate_key() (*Wallet method*), 105
genesis() (*AlgodClient method*), 98
genesis_hash (*ApplicationCallTxn attribute*), 60
genesis_hash (*AssetConfigTxn attribute*), 53, 86
genesis_hash (*AssetFreezeTxn attribute*), 56, 88
genesis_hash (*AssetTransferTxn attribute*), 58, 89
genesis_hash (*KeyregNonparticipatingTxn attribute*), 52
genesis_hash (*KeyregOfflineTxn attribute*), 51
genesis_hash (*KeyregOnlineTxn attribute*), 50
genesis_hash (*KeyregTxn attribute*), 48, 84
genesis_hash (*PaymentTxn attribute*), 47, 82
genesis_id (*AssetConfigTxn attribute*), 54, 86
genesis_id (*AssetFreezeTxn attribute*), 57, 88
genesis_id (*AssetTransferTxn attribute*), 58, 89
genesis_id (*KeyregNonparticipatingTxn attribute*), 52
genesis_id (*KeyregOfflineTxn attribute*), 51
genesis_id (*KeyregOnlineTxn attribute*), 50
genesis_id (*KeyregTxn attribute*), 48, 84
genesis_id (*PaymentTxn attribute*), 47, 82
get_address() (*Template method*), 41, 77

get_application_address() (in module *algorand.logic*), 76
 get_mnemonic() (Wallet method), 105
 get_multisig_account() (Multisig method), 67, 91
 get_program() (DynamicFee method), 43, 79
 get_program() (HTLC method), 42, 79
 get_program() (LimitOrder method), 44, 81
 get_program() (PeriodicPayment method), 44, 80
 get_program() (Split method), 42, 78
 get_program() (Template method), 41, 77
 get_public_keys() (Multisig method), 67, 91
 get_selector() (Method method), 22
 get_signature() (Method method), 22
 get_split_funds_transaction() (Split static method), 42, 78
 get_status() (AtomicTransactionComposer method), 30
 get_swap_assets_transactions() (LimitOrder static method), 44, 81
 get_transaction() (HTLC static method), 43, 79
 get_transactions() (DynamicFee static method), 43, 79
 get_tx_count() (AtomicTransactionComposer method), 30
 get_txid() (LogicSigTransaction method), 70
 get_txid() (MultisigTransaction method), 66
 get_txid() (SignedTransaction method), 65
 get_txid() (Transaction method), 46, 81
 get_txn_calls() (Method method), 22
 get_wallet() (KMDClient method), 72
 get_withdrawal_transaction() (PeriodicPayment static method), 44, 80
 gh (SuggestedParams attribute), 45
 global_schema (ApplicationCallTxn attribute), 61
 group (KeyregNonparticipatingTxn attribute), 52
 group (KeyregOfflineTxn attribute), 51
 group (KeyregOnlineTxn attribute), 50
 group (KeyregTxn attribute), 48, 84
 group (PaymentTxn attribute), 47, 82
 import_multisig() (KMDClient method), 74
 import_multisig() (Wallet method), 106
 index (ApplicationCallTxn attribute), 60
 index (AssetConfigTxn attribute), 53, 86
 index (AssetFreezeTxn attribute), 56, 88
 index (AssetTransferTxn attribute), 58, 89
 indexer_address (IndexerClient attribute), 98
 INDEXER_AUTH_HEADER (in module *algorand.constants*), 35
 indexer_request() (IndexerClient method), 98
 indexer_token (IndexerClient attribute), 98
 IndexerClient (class in *algorand.v2client.indexer*), 98
 IndexerHTTPError, 41
 info() (Wallet method), 105
 init_handle() (Wallet method), 107
 init_wallet_handle() (KMDClient method), 72
 inject() (in module *algorand.future.template*), 45
 inject() (in module *algorand.template*), 81
 int_list() (ApplicationCallTxn static method), 61
 Interface (class in *algorand.abi.interface*), 21
 InvalidProgram, 40
 InvalidSecretKeyError, 39
 InvalidThresholdError, 39
 is_abi_reference_type() (in module *algorand.abi.reference*), 23
 is_abi_transaction_type() (in module *algorand.abi.transaction*), 24
 is_delegated() (LogicSigAccount method), 68
 is_dynamic() (ABIType method), 19
 is_dynamic() (AddressType method), 17
 is_dynamic() (ArrayDynamicType method), 18
 is_dynamic() (ArrayStaticType method), 19
 is_dynamic() (BoolType method), 20
 is_dynamic() (ByteType method), 20
 is_dynamic() (StringType method), 23
 is_dynamic() (TupleType method), 24
 is_dynamic() (UfixedType method), 25
 is_dynamic() (UintType method), 25
 is_valid_address() (in module *algorand.encoding*), 39

H

handle (Wallet attribute), 105
 HASH_LEN (in module *algorand.constants*), 37
 headers (AlgodClient attribute), 26, 95
 headers (IndexerClient attribute), 98
 health() (AlgodClient method), 27, 97
 health() (IndexerClient method), 99
 HTLC (class in *algorand.future.template*), 42
 HTLC (class in *algorand.template*), 78

I

id (Wallet attribute), 105
 import_key() (KMDClient method), 73
 import_key() (Wallet method), 105

J

json_dictify() (Multisig method), 67, 91
 json_dictify() (MultisigSubsig method), 67, 92

K

kcl (Wallet attribute), 105
 KEN_LEN_BYTES (in module *algorand.constants*), 37
 KEYREG (ABITransactionType attribute), 23
 KEYREG_TXN (in module *algorand.constants*), 36
 KeyregNonparticipatingTxn (class in *algorand.future.transaction*), 51
 KeyregOfflineTxn (class in *algorand.future.transaction*), 50

KeyregOnlineTxn (*class in* algosdk.future.transaction*),* 49
 KeyregOnlineTxnInitError, 41
 KeyregTxn (*class in* algosdk.future.transaction*),* 47
 KeyregTxn (*class in* algosdk.transaction*),* 83
 kmd_address (*KMDClient attribute*), 71
 KMD_AUTH_HEADER (*in module* algosdk.constants*),* 35
 kmd_request() (*KMDClient method*), 71
 kmd_token (*KMDClient attribute*), 71
 KMDClient (*class in* algosdk.kmd*),* 71
 KMDHTTPError, 41

L

last (*SuggestedParams attribute*), 45
 last_valid_round (*ApplicationCallTxn attribute*), 60
 last_valid_round (*AssetConfigTxn attribute*), 53, 86
 last_valid_round (*AssetFreezeTxn attribute*), 56, 87
 last_valid_round (*AssetTransferTxn attribute*), 58, 89
 last_valid_round (*KeyregNonparticipatingTxn attribute*), 52
 last_valid_round (*KeyregOfflineTxn attribute*), 51
 last_valid_round (*KeyregOnlineTxn attribute*), 50
 last_valid_round (*KeyregTxn attribute*), 48, 84
 last_valid_round (*PaymentTxn attribute*), 47, 82
 lease (*AssetConfigTxn attribute*), 54, 87
 lease (*AssetFreezeTxn attribute*), 57, 88
 lease (*AssetTransferTxn attribute*), 58, 89
 lease (*KeyregNonparticipatingTxn attribute*), 52
 lease (*KeyregOfflineTxn attribute*), 51
 lease (*KeyregOnlineTxn attribute*), 50
 lease (*KeyregTxn attribute*), 49, 84
 lease (*PaymentTxn attribute*), 47, 83
 LEASE_LENGTH (*in module* algosdk.constants*),* 38
 ledger_supply() (*AlgodClient method*), 27, 96
 LimitOrder (*class in* algosdk.future.template*),* 44
 LimitOrder (*class in* algosdk.template*),* 80
 list_assets() (*AlgodClient method*), 28
 list_keys() (*KMDClient method*), 74
 list_keys() (*Wallet method*), 105
 list_multisig() (*KMDClient method*), 74
 list_multisig() (*Wallet method*), 106
 list_wallets() (*KMDClient method*), 72
 local_schema (*ApplicationCallTxn attribute*), 61
 logic (*LogicSig attribute*), 67, 92
 LOGIC_DATA_PREFIX (*in module* algosdk.constants*),* 37
 LOGIC_PREFIX (*in module* algosdk.constants*),* 37
 LOGIC_SIG_MAX_COST (*in module* algosdk.constants*),* 38
 LOGIC_SIG_MAX_SIZE (*in module* algosdk.constants*),* 38
 LogicSig (*class in* algosdk.future.transaction*),* 67
 LogicSig (*class in* algosdk.transaction*),* 92
 LogicSigAccount (*class in* algosdk.future.transaction*),* 68
 LogicSigOverspecifiedSignature, 40
 LogicSigSigningKeyMissing, 40

LogicSigTransaction (*class in* algosdk.future.transaction*),* 69
 LogicSigTransaction (*class in* algosdk.transaction*),* 93
 LogicSigTransactionSigner (*class in* algosdk.atomic_transaction_composer*),* 33
 lsig (*LogicSigTransaction attribute*), 69, 93

M

manager (*AssetConfigTxn attribute*), 54, 86
 MAX_APP_ARG_LIMIT (*AtomicTransactionComposer attribute*), 30
 MAX_ASSET_DECIMALS (*in module* algosdk.constants*),* 38
 MAX_GROUP_SIZE (*AtomicTransactionComposer attribute*), 30
 max_price (*Bid attribute*), 34
 merge() (*MultisigTransaction static method*), 66, 91
 MergeAuthAddrMismatchError, 39
 MergeKeysMismatchError, 39
 metadata_hash (*AssetConfigTxn attribute*), 54, 86
 METADATA_LENGTH (*in module* algosdk.constants*),* 37
 Method (*class in* algosdk.abi.method*),* 22
 microalgos_to_algos() (*in module* algosdk.util*),* 94
 MICROALGOS_TO_ALGOS_RATIO (*in module* algosdk.constants*),* 37
 min_fee (*SuggestedParams attribute*), 45
 MIN_TXN_FEE (*in module* algosdk.constants*),* 37
 MNEMONIC_LEN (*in module* algosdk.constants*),* 37
 module
 algosdk.abi.address_type, 17
 algosdk.abi.array_dynamic_type, 18
 algosdk.abi.array_static_type, 18
 algosdk.abi.base_type, 19
 algosdk.abi.bool_type, 20
 algosdk.abi.byte_type, 20
 algosdk.abi.contract, 21
 algosdk.abi.interface, 21
 algosdk.abi.method, 22
 algosdk.abi.reference, 23
 algosdk.abi.string_type, 23
 algosdk.abi.transaction, 23
 algosdk.abi.tuple_type, 24
 algosdk.abi.ufixed_type, 25
 algosdk.abi.uint_type, 25
 algosdk.account, 26
 algosdk.algod, 26
 algosdk.atomic_transaction_composer, 29
 algosdk.auction, 34
 algosdk.constants, 35
 algosdk.encoding, 38
 algosdk.error, 39
 algosdk.future.template, 41
 algosdk.future.transaction, 45
 algosdk.kmd, 71

algosdk.logic, 75
 algosdk.mnemonic, 77
 algosdk.template, 77
 algosdk.transaction, 81
 algosdk.util, 94
 algosdk.v2client.algod, 95
 algosdk.v2client.indexer, 98
 algosdk.wallet, 104
 algosdk.wordlist, 107
 msgpack_decode() (in module algosdk.encoding), 39
 msgpack_encode() (in module algosdk.encoding), 38
 msig (LogicSig attribute), 67, 92
 MSIG_ADDR_PREFIX (in module algosdk.constants), 37
 Multisig (class in algosdk.future.transaction), 66
 Multisig (class in algosdk.transaction), 91
 multisig (MultisigTransaction attribute), 65, 90
 MULTISIG_ACCOUNT_LIMIT (in module algosdk.constants), 38
 MultisigAccountSizeError, 40
 MultisigSubsig (class in algosdk.future.transaction), 67
 MultisigSubsig (class in algosdk.transaction), 91
 MultisigTransaction (class in algosdk.future.transaction), 65
 MultisigTransaction (class in algosdk.transaction), 90
 MultisigTransactionSigner (class in algosdk.atomic_transaction_composer), 33

N

name (Wallet attribute), 104
 NetworkInfo (class in algosdk.abi.contract), 21
 new_freeze_state (AssetFreezeTxn attribute), 56, 88
 NO_AUTH (in module algosdk.constants), 36
 nonpart (KeyregTxn attribute), 49
 NoOpOC (OnComplete attribute), 59
 note (AssetConfigTxn attribute), 54, 86
 note (AssetFreezeTxn attribute), 57, 88
 note (AssetTransferTxn attribute), 58, 89
 note (KeyregNonparticipatingTxn attribute), 52
 note (KeyregOfflineTxn attribute), 51
 note (KeyregOnlineTxn attribute), 50
 note (KeyregTxn attribute), 48, 84
 note (PaymentTxn attribute), 47, 82
 note_field_type (NoteField attribute), 35
 NOTE_FIELD_TYPE_BID (in module algosdk.constants), 36
 NOTE_FIELD_TYPE_DEPOSIT (in module algosdk.constants), 36
 NOTE_FIELD_TYPE_PARAMS (in module algosdk.constants), 36
 NOTE_FIELD_TYPE_SETTLEMENT (in module algosdk.constants), 36
 NOTE_MAX_LENGTH (in module algosdk.constants), 38

NoteField (class in algosdk.auction), 35
 num_byte_slices (StateSchema attribute), 59
 num_uints (StateSchema attribute), 59

O

on_complete (ApplicationCallTxn attribute), 60
 OnComplete (class in algosdk.future.transaction), 59
 OptInOC (OnComplete attribute), 59
 OutOfRangeDecimalsError, 40
 OverspecifiedRoundError, 40

P

parse_uvarint() (in module algosdk.logic), 76
 PAY (ABITransactionType attribute), 23
 PAYMENT_TXN (in module algosdk.constants), 36
 PaymentTxn (class in algosdk.future.transaction), 46
 PaymentTxn (class in algosdk.transaction), 82
 pending_transaction_info() (AlgodClient method), 28, 97
 pending_transactions() (AlgodClient method), 27, 97
 pending_transactions_by_address() (AlgodClient method), 96
 PeriodicPayment (class in algosdk.future.template), 43
 PeriodicPayment (class in algosdk.template), 80
 populate_foreign_array() (in module algosdk.atomic_transaction_composer), 29
 precision (UfixedType attribute), 25
 proof() (AlgodClient method), 98
 pswd (Wallet attribute), 104
 public_key (MultisigSubsig attribute), 67, 91
 put_uvarint() (in module algosdk.future.template), 45
 put_uvarint() (in module algosdk.template), 81

R

raw_sign() (Transaction method), 46, 81
 read_byte_const_block() (in module algosdk.logic), 76
 read_int_const_block() (in module algosdk.logic), 76
 read_program() (in module algosdk.logic), 76
 read_push_byte_block() (in module algosdk.logic), 76
 read_push_int_block() (in module algosdk.logic), 76
 receiver (AssetTransferTxn attribute), 58, 89
 receiver (PaymentTxn attribute), 47, 82
 rekey (AssetConfigTxn attribute), 54
 rekey_to (AssetConfigTxn attribute), 87
 rekey_to (AssetFreezeTxn attribute), 57, 88
 rekey_to (AssetTransferTxn attribute), 58, 90
 rekey_to (KeyregNonparticipatingTxn attribute), 52
 rekey_to (KeyregOfflineTxn attribute), 51
 rekey_to (KeyregOnlineTxn attribute), 50
 rekey_to (KeyregTxn attribute), 49, 84

rekey_to (*PaymentTxn* attribute), 47, 83
 release_handle() (*Wallet* method), 107
 release_wallet_handle() (*KMDClient* method), 72
 rename() (*Wallet* method), 105
 rename_wallet() (*KMDClient* method), 73
 renew_handle() (*Wallet* method), 107
 renew_wallet_handle() (*KMDClient* method), 73
 required() (*Transaction* static method), 46
 reserve (*AssetConfigTxn* attribute), 54, 86
 retrieve_from_file() (in module *al-gosdk.future.transaction*), 70
 retrieve_from_file() (in module *al-gosdk.transaction*), 93
 Returns (class in *algosdk.abi.method*), 22
 revocation_target (*AssetTransferTxn* attribute), 58, 89

S

search_applications() (*IndexerClient* method), 103
 search_asset_transactions() (*IndexerClient* method), 102
 search_assets() (*IndexerClient* method), 103
 search_transactions() (*IndexerClient* method), 100
 search_transactions_by_address() (*IndexerClient* method), 101
 selkey (*KeyregOnlineTxn* attribute), 50
 selkey (*KeyregTxn* attribute), 48, 84
 send_raw_transaction() (*AlgodClient* method), 28, 96
 send_transaction() (*AlgodClient* method), 28, 96
 send_transactions() (*AlgodClient* method), 29, 97
 sender (*ApplicationCallTxn* attribute), 60
 sender (*AssetConfigTxn* attribute), 53, 85
 sender (*AssetFreezeTxn* attribute), 56, 87
 sender (*AssetTransferTxn* attribute), 57, 89
 sender (*KeyregNonparticipatingTxn* attribute), 52
 sender (*KeyregOfflineTxn* attribute), 51
 sender (*KeyregOnlineTxn* attribute), 49
 sender (*KeyregTxn* attribute), 48, 83
 sender (*PaymentTxn* attribute), 47, 82
 sig (*LogicSig* attribute), 67, 92
 sign() (*Bid* method), 34
 sign() (*LogicSig* method), 68, 92
 sign() (*LogicSigAccount* method), 69
 sign() (*MultisigTransaction* method), 65, 90
 sign() (*Transaction* method), 46, 81
 sign_bytes() (in module *algosdk.util*), 94
 sign_dynamic_fee() (*DynamicFee* method), 43, 79
 sign_multisig() (*LogicSigAccount* method), 69
 sign_multisig_transaction() (*KMDClient* method), 75
 sign_multisig_transaction() (*Wallet* method), 106
 sign_program() (*LogicSig* static method), 68, 92
 sign_transaction() (*KMDClient* method), 74

sign_transaction() (*Wallet* method), 106
 sign_transactions() (*AccountTransactionSigner* method), 33
 sign_transactions() (*LogicSigTransactionSigner* method), 33
 sign_transactions() (*MultisigTransactionSigner* method), 33
 sign_transactions() (*TransactionSigner* method), 32
 signature (*MultisigSubsig* attribute), 67, 92
 signature (*SignedBid* attribute), 35
 signature (*SignedTransaction* attribute), 65, 90
 SIGNED (*AtomicTransactionComposerStatus* attribute), 29
 signed_bid (*NoteField* attribute), 35
 SignedBid (class in *algosdk.auction*), 35
 SignedTransaction (class in *al-gosdk.future.transaction*), 65
 SignedTransaction (class in *algosdk.transaction*), 90
 single_sig_multisig() (*LogicSig* static method), 68, 92
 Split (class in *algosdk.future.template*), 41
 Split (class in *algosdk.template*), 77
 sprfkey (*KeyregOnlineTxn* attribute), 50
 sprfkey (*KeyregTxn* attribute), 49
 state_schema() (*ApplicationCallTxn* static method), 61
 StateSchema (class in *algosdk.future.transaction*), 59
 static_length (*ArrayStaticType* attribute), 18
 status() (*AlgodClient* method), 27, 96
 status_after_block() (*AlgodClient* method), 27, 96
 StringType (class in *algosdk.abi.string_type*), 23
 submit() (*AtomicTransactionComposer* method), 32
 SUBMITTED (*AtomicTransactionComposerStatus* attribute), 29
 subsigs (*Multisig* attribute), 66, 91
 suggested_fee() (*AlgodClient* method), 28
 suggested_params() (*AlgodClient* method), 28, 97
 suggested_params_as_object() (*AlgodClient* method), 28
 SuggestedParams (class in *algosdk.future.transaction*), 45

T

target (*AssetFreezeTxn* attribute), 56, 88
 teal_bytes() (*ApplicationCallTxn* static method), 61
 teal_sign() (in module *algosdk.logic*), 76
 teal_sign_from_program() (in module *al-gosdk.logic*), 76
 Template (class in *algosdk.future.template*), 41
 Template (class in *algosdk.template*), 77
 TemplateError, 41
 TemplateInputError, 41
 TGID_PREFIX (in module *algosdk.constants*), 36
 threshold (*Multisig* attribute), 66, 91

to_master_derivation_key() (in module *algorithmsdk.mnemonic*), 77
 to_private_key() (in module *algorithmsdk.mnemonic*), 77
 to_public_key() (in module *algorithmsdk.mnemonic*), 77
 total (*AssetConfigTxn* attribute), 53, 86
 Transaction (class in *algorithmsdk.future.transaction*), 45
 Transaction (class in *algorithmsdk.transaction*), 81
 transaction (*LogicSigTransaction* attribute), 69, 93
 transaction (*MultisigTransaction* attribute), 65, 90
 transaction (*SignedTransaction* attribute), 65, 90
 transaction() (*IndexerClient* method), 100
 transaction_by_id() (*AlgodClient* method), 28
 transaction_info() (*AlgodClient* method), 28
 TransactionGroupSizeError, 40
 transactions_by_address() (*AlgodClient* method), 27
 TransactionSigner (class in *algorithmsdk.atomic_transaction_composer*), 32
 TransactionWithSigner (class in *algorithmsdk.atomic_transaction_composer*), 33
 TupleType (class in *algorithmsdk.abi.tuple_type*), 24
 TX_GROUP_LIMIT (in module *algorithmsdk.constants*), 38
 TxGroup (class in *algorithmsdk.future.transaction*), 70
 TxGroup (class in *algorithmsdk.transaction*), 93
 TXID_PREFIX (in module *algorithmsdk.constants*), 36
 type (*AssetConfigTxn* attribute), 54, 86
 type (*AssetFreezeTxn* attribute), 57, 88
 type (*AssetTransferTxn* attribute), 58, 89
 type (*KeyregNonparticipatingTxn* attribute), 52
 type (*KeyregOfflineTxn* attribute), 51
 type (*KeyregOnlineTxn* attribute), 50
 type (*KeyregTxn* attribute), 49, 84
 type (*PaymentTxn* attribute), 47, 83

U

UfixedType (class in *algorithmsdk.abi.ufixed_type*), 25
 UintType (class in *algorithmsdk.abi.uint_type*), 25
 UnderspecifiedRoundError, 41
 undictify() (*Argument* static method), 22
 undictify() (*Bid* static method), 35
 undictify() (*Contract* static method), 21
 undictify() (*Interface* static method), 21
 undictify() (*LogicSig* static method), 67, 92
 undictify() (*LogicSigAccount* static method), 68
 undictify() (*LogicSigTransaction* static method), 70, 93
 undictify() (*Method* static method), 22
 undictify() (*Multisig* static method), 67, 91
 undictify() (*MultisigSubsig* static method), 67, 92
 undictify() (*MultisigTransaction* static method), 66, 90
 undictify() (*NetworkInfo* static method), 21
 undictify() (*NoteField* static method), 35
 undictify() (*Returns* static method), 23

undictify() (*SignedBid* static method), 35
 undictify() (*SignedTransaction* static method), 65, 90
 undictify() (*StateSchema* static method), 59
 undictify() (*Transaction* static method), 46, 81
 undictify() (*TxGroup* static method), 70, 94
 unit_name (*AssetConfigTxn* attribute), 54, 86
 UnknownMsigVersionError, 40
 UNVERSIONED_PATHS (in module *algorithmsdk.constants*), 36
 UpdateApplicationOC (*OnComplete* attribute), 59
 url (*AssetConfigTxn* attribute), 54, 86

V

validate() (*Multisig* method), 66, 91
 verify() (*LogicSig* method), 67, 92
 verify() (*LogicSigAccount* method), 68
 verify() (*LogicSigTransaction* method), 70, 93
 verify() (*Multisig* method), 67, 91
 verify_bytes() (in module *algorithmsdk.util*), 94
 version (*Multisig* attribute), 66, 91
 versions() (*AlgodClient* method), 27, 97
 versions() (*KMDCClient* method), 72
 VOID (*Returns* attribute), 22
 votefst (*KeyregOnlineTxn* attribute), 50
 votefst (*KeyregTxn* attribute), 49, 84
 votekd (*KeyregOnlineTxn* attribute), 50
 votekd (*KeyregTxn* attribute), 49, 84
 votelst (*KeyregOnlineTxn* attribute), 50
 votelst (*KeyregTxn* attribute), 49, 84
 voteapk (*KeyregOnlineTxn* attribute), 50
 voteapk (*KeyregTxn* attribute), 48, 84

W

wait_for_confirmation() (in module *algorithmsdk.future.transaction*), 71
 Wallet (class in *algorithmsdk.wallet*), 104
 word_list_raw() (in module *algorithmsdk.wordlist*), 107
 write_to_file() (in module *algorithmsdk.future.transaction*), 70
 write_to_file() (in module *algorithmsdk.transaction*), 93
 WrongAmountType, 40
 WrongChecksumError, 40
 WrongContractError, 40
 WrongHashLengthError, 40
 WrongKeyBytesLengthError, 40
 WrongKeyLengthError, 40
 WrongLeaseLengthError, 40
 WrongMetadataLengthError, 40
 WrongMnemonicLengthError, 40
 WrongNoteLength, 40
 WrongNoteType, 40

Z

ZeroAddressError, 41