
aioeos

Maciej Janiszewski

Apr 22, 2020

CONTENTS

1	Introduction	1
1.1	Features	1
1.2	Missing features	1
1.3	Getting Started	2
1.3.1	Running your testnet	2
1.3.2	Submitting your first transaction	3
1.3.3	Example code	4
2	API	5
2.1	Contracts	5
2.1.1	eosio	5
2.1.2	eosio_token	5
2.2	Exceptions	5
2.3	Keys	6
2.4	RPC	6
2.5	Serializer	6
2.6	Types	9
3	Changelog	13
3.1	1.0.2 (10.04.2020)	13
3.2	1.0.1 (03.04.2020)	13
3.3	1.0.0 (01.04.2020)	14
4	Indices and tables	15
	Python Module Index	17
	Index	19

INTRODUCTION

aioeos is an `ssync` Python library for interacting with EOS.io blockchain. Library consists of an async wrapper for [Nodeos RPC API](#), a serializer for basic ABI types like transactions and actions and private key management. Helpers for generating actions such as creating new accounts, buying and selling RAM etc. can be imported from `aioeos.contracts` namespace.

Please bear in mind that the serializer is not complete. Action payloads need to be converted to binary format using `/abi_json_to_bin` endpoint on the RPC node. Use only nodes you trust.

1.1 Features

1. Async JSON-RPC client.
2. Signing and verifying transactions using private and public keys.
3. Serializer for basic EOS.io blockchain ABI types.
4. Helpers which provide an easy way to generate common actions, such as token transfer.

1.2 Missing features

1. Serializer and deserializer for action payloads.
2. Support for types:
 - `bool`,
 - `uint128`,
 - `int128`,
 - `float128`,
 - `block_timestamp_type`,
 - `symbol`,
 - `symbol_code`,
 - `asset`,
 - `checksum160`,
 - `checksum256`,
 - `checksum512`,

- `public_key`,
- `private_key`,
- `signature`,
- `extended_asset`

1.3 Getting Started

This guide step-by-step explains how to use aioeos library to submit your first transaction. Complete example is available at the end of this chapter. Before we begin, please make sure you have `cleos` utility installed in your system (part of `eosio` package) and that `aioeos` is installed.

On macOS:

```
$ brew install eosio
$ pip install aioeos
```

1.3.1 Running your testnet

Along with the library, we provide an EOS testnet Docker image. Due to [this issue](#) we recommend cloning the `eos-testnet` repository and running `ensure_eosio.sh` script.

```
$ git clone https://github.com/ulamlabs/eos-testnet.git
$ cd eos-testnet
$ ./ensure_eosio.sh
# You can check that server is running
$ cleos get info
```

Image by default comes with a hardcoded test account:

- **Account name:** `eostest12345`
- **Private key:** `5JeaxignXEg3mGwvgnmxG6w6wHcRp9ooPw81KjrP2ah6TWSECDN`
- **Public key:** `EOS8VhvYTcUMwp9jFD8UWRMPgWsGQoqBfpBvrjjfMCouqRH9JF5qW`

You can parametrize this through `env` variables, please refer to the [Docker image README](#).

Let's create another account to send funds to.

```
# If you don't have a wallet yet, otherwise open it and unlock
$ cleos wallet create -f ~/.eosio-wallet-pass

# Import keys for eostest12345 account
$ cleos wallet import --private-key 
↪5JeaxignXEg3mGwvgnmxG6w6wHcRp9ooPw81KjrP2ah6TWSECDN

# Create your second account, for example mysecondacc1
$ cleos system newaccount eostest12345 --transfer mysecondacc1 
↪EOS8VhvYTcUMwp9jFD8UWRMPgWsGQoqBfpBvrjjfMCouqRH9JF5qW --stake-net "1.0000 EOS" --
↪stake-cpu "1.0000 EOS" --buy-ram-kbytes 8192
```

1.3.2 Submitting your first transaction

Let's serialize and submit a basic transaction to EOS.io blockchain. We can think about a transaction as a set of contract calls that we want to execute. These are called actions. Along with the action itself, we provide a list of authorizations. These are defined per action. It basically tells the blockchain which keys will be used to sign this transaction.

Let's say we want to transfer 1.0000 EOS from *eostest12345* to *mysecondacc1* account.

```
from aioeos import EosAccount, EosTransaction
from aioeos.contracts import eosio_token

test_account = EosAccount(
    name='eostest12345',
    private_key='5JeaxignXEg3mGwvgmwxG6w6wHcRp9ooPw81KjrP2ah6TWSECDN'
)

action = eosio_token.transfer(
    from_addr=test_account.name,
    to_addr='mysecondacc1',
    quantity='1.0000 EOS',
    authorization=[
        test_account.authorization('active')
    ]
)
```

Let's also create an instance of *EosJsonRpc*. Remember to always **USE ONLY NODES THAT YOU TRUST**. Because aioeos doesn't currently support serialization of action payloads, for this transaction to be ready to be submitted to the blockchain, we need to ask our RPC node to convert it for us.

```
from aioeos import EosJsonRpc

rpc = EosJsonRpc(url='http://127.0.0.1:8888')
```

Now, let's create a transaction containing this action. Each transaction needs to contain TAPOS fields. These tell the EOS.io blockchain when the transaction is considered valid, such as the first block in which it can be included, as well as an expiration date. While we can provide those parameters manually if we want to, we can also use the RPC to find out the right block number and prefix. Let's assume that we want these transaction to be valid for next 2 minutes.

```
from datetime import datetime, timedelta

block = await rpc.get_head_block()
transaction = EosTransaction(
    expiration=datetime.now() + timedelta(minutes=2)
    ref_block_num=block['block_num'] & 65535,
    ref_block_prefix=block['ref_block_prefix'],
    actions=[action]
)
```

Transaction is now ready to be submitted to the blockchain. It's time to serialize, sign and push it. An EOS transaction signature is a digest of the following data:

- Chain ID - identifies the blockchain that transaction is submitted against,
- Transaction,
- 32 context-free bytes - these can be left empty in this case

While we could do it manually, RPC client provides a helper method which does all of that for us.

```
response = await rpc.sign_and_push_transaction(  
    transaction, keys=[test_account.key]  
)
```

1.3.3 Example code

Complete example code:

```
import asyncio  
  
from aioeos import EosAccount, EosJsonRpc, EosTransaction  
from aioeos.contracts import eosio_token  
  
async def example():  
    test_account = EosAccount(  
        name='eostest12345',  
        private_key='5JeaxignXEg3mGwvqmwG6w6wHcRp9ooPw81Kjrp2ah6TWSECDN'  
    )  
  
    action = eosio_token.transfer(  
        from_addr=test_account.name,  
        to_addr='mysecondacc1',  
        quantity='1.0000 EOS',  
        authorization=[test_account.authorization('active')]  
    )  
  
    rpc = EosJsonRpc(url='http://127.0.0.1:8888')  
    block = await rpc.get_head_block()  
  
    transaction = EosTransaction(  
        ref_block_num=block['block_num'] & 65535,  
        ref_block_prefix=block['ref_block_prefix'],  
        actions=[action]  
    )  
  
    response = await rpc.sign_and_push_transaction(  
        transaction, keys=[test_account.key]  
    )  
    print(response)  
  
asyncio.get_event_loop().run_until_complete(example())
```


2.1 Contracts

2.1.1 eosio

2.1.2 eosio_token

2.2 Exceptions

exception `aioeos.exceptions.EosAccountDoesntExistException`

Thrown by `get_account` where account doesn't exist

exception `aioeos.exceptions.EosAccountExistsException`

Thrown by `create_wallet` where account with given name already exists

exception `aioeos.exceptions.EosActionValidateException`

Raised when action payload is invalid

exception `aioeos.exceptions.EosAssertMessageException`

Generic assertion error from smart contract, can mean literally anything, need to parse C++ traceback to figure out what went wrong.

exception `aioeos.exceptions.EosDeadlineException`

Transaction timed out

exception `aioeos.exceptions.EosMissingTaposFieldsException`

TAPOS fields are missing from Transaction object

exception `aioeos.exceptions.EosRamUsageExceededException`

Transaction requires more RAM than what's available on the account

exception `aioeos.exceptions.EosRpcException`

Base EOS exception

exception `aioeos.exceptions.EosSerializerAbiNameInvalidCharactersException`

exception `aioeos.exceptions.EosSerializerAbiNameTooLongException`

exception `aioeos.exceptions.EosSerializerException`

Base exception class for serializer errors

exception `aioeos.exceptions.EosSerializerUnsupportedTypeException`

Our serializer doesn't support provided object type

exception `aioeos.exceptions.EosTxCpuUsageExceededException`

Not enough EOS were staked for CPU

exception `aioeos.exceptions.EosTxNetUsageExceededException`
Not enough EOS were staked for NET

2.3 Keys

class `aioeos.keys.EosKey` (*, *private_key=None*, *public_key=None*)
EosKey instance.

Depends on which kwargs are given, this works in a different way: - No kwargs - generates a new private key
- Only *private_key* - public key is being derived from private key - Only *public_key* - EosKey instance has no private key

sign (*digest*)

Signs sha256 hash with private key. Returns signature in format: SIG_K1_{digest}

to_key_weight (*weight*)

Return type EosKeyWeight

to_public ()

Returns compressed, base58 encoded public key prefixed with EOS

to_pvt (*key_type='K1'*)

Converts private key to PVT format

to_wif ()

Converts private key to legacy WIF format

verify (*encoded_sig*, *digest*)

Verifies signature with private key

Return type bool

2.4 RPC

2.5 Serializer

class `aioeos.serializer.AbiActionPayloadSerializer`

deserialize (*value*)

Returns a tuple containing length of original data and deserialized value

Return type Tuple[int, bytes]

serialize (*value*)

Returns byte-encoded value

Return type bytes

class `aioeos.serializer.AbiBytesSerializer`

Serializer for ABI bytes type. Serialized value consists of raw bytes prefixed with payload size encoded as VarUInt.

deserialize (*value*)

Returns a tuple containing length of original data and deserialized value

Return type Tuple[int, bytes]

serialize (*value*)
Returns byte-encoded value

Return type bytes

class aioeos.serializer.**AbiListSerializer** (*list_type*)
Serializer for ABI List type. In binary format, it basically looks like this: [count][item 1][item 2]...

deserialize (*value*)
Returns a tuple containing length of original data and deserialized value

Return type Tuple[int, List[Any]]

serialize (*value*)
Returns byte-encoded value

Return type bytes

class aioeos.serializer.**AbiNameSerializer**
Serializer for ABI names. ABI names can only contain these characters: .
1234567890abcdefghijklmnopqrstuvwxyz. Maximum length is 13 chars.

deserialize (*value*)
Returns a tuple containing length of original data and deserialized value

Return type Tuple[int, str]

serialize (*value*)
Returns byte-encoded value

Return type bytes

class aioeos.serializer.**AbiObjectSerializer** (*abi_class*)

deserialize (*value*)
Returns a tuple containing length of original data and deserialized value

Return type Tuple[int, BaseAbiObject]

serialize (*value*)
Returns byte-encoded value

Return type bytes

class aioeos.serializer.**AbiStringSerializer**
Serializer for ABI String type. String format is similar to bytes as it's prefixed with length but it's comprised of ASCII codes for each character packed in binary format.

deserialize (*value*)
Returns a tuple containing length of original data and deserialized value

Return type Tuple[int, str]

serialize (*value*)
Returns byte-encoded value

Return type bytes

class aioeos.serializer.**AbiTimePointSecSerializer**
Serializer for ABI TimePointSec type. It's essentially a timestamp.

deserialize (*value*)
Returns a tuple containing length of original data and deserialized value

Return type Tuple[int, datetime]

serialize (*value*)
Returns byte-encoded value

Return type bytes

class aioeos.serializer.**AbiTimePointSerializer**
Serializer for ABI TimePoint type. Encodes timestamp with milisecond precision.

deserialize (*value*)
Returns a tuple containing length of original data and deserialized value

Return type Tuple[int, datetime]

serialize (*value*)
Returns byte-encoded value

Return type bytes

class aioeos.serializer.**BaseSerializer**

abstract deserialize (*value*)
Returns a tuple containing length of original data and deserialized value

Return type Tuple[int, Any]

abstract serialize (*value*)
Returns byte-encoded value

Return type bytes

class aioeos.serializer.**BasicTypeSerializer** (*fmt=""*)
Serializes basic types such as integers and floats using `struct` module

Params *fmt* format string, please refer to documentation for [struct module](#)

deserialize (*value*)
Returns a tuple containing length of original data and deserialized value

Return type Tuple[int, Any]

serialize (*value*)
Returns byte-encoded value

Return type bytes

class aioeos.serializer.**VarUIntSerializer**
Serializer for ABI VarUInt type. This type has different length based on how many bytes are required to encode given integer.

deserialize (*value*)
Returns a tuple containing length of original data and deserialized value

Return type Tuple[int, int]

serialize (*value*)
Returns byte-encoded value

Return type bytes

aioeos.serializer.**deserialize** (*value*, *abi_class*)
Deserializes ABI values from binary format

Return type Tuple[int, Any]

`aioeos.serializer.get_abi_type_serializer` (*abi_type*)

Return type `BaseSerializer`

`aioeos.serializer.serialize` (*value*, *abi_type=None*)

Serializes ABI values to binary format

Return type `bytes`

2.6 Types

`aioeos.types.UInt8`

alias of `builtins.int`

`aioeos.types.UInt16`

alias of `builtins.int`

`aioeos.types.UInt32`

alias of `builtins.int`

`aioeos.types.UInt64`

alias of `builtins.int`

`aioeos.types.Int8`

alias of `builtins.int`

`aioeos.types.Int16`

alias of `builtins.int`

`aioeos.types.Int32`

alias of `builtins.int`

`aioeos.types.Int64`

alias of `builtins.int`

`aioeos.types.VarUInt`

alias of `builtins.int`

`aioeos.types.Float32`

alias of `builtins.float`

`aioeos.types.Float64`

alias of `builtins.float`

`aioeos.types.TimePointSec`

alias of `datetime.datetime`

`aioeos.types.TimePoint`

alias of `datetime.datetime`

`aioeos.types.Name`

alias of `builtins.str`

`aioeos.types.AbiBytes`

alias of `builtins.bytes`

class `aioeos.types.BaseAbiObject`

`aioeos.types.is_abi_object` (*obj*)

Object is an ABI object if it's a subclass of `BaseAbiObject`

Return type `bool`

```
class aioeos.types.EosPermissionLevel (actor, permission)

    actor: Name = None
    permission: Name = None
class aioeos.types.EosKeyWeight (key, weight)

    key: AbiBytes = None
    weight: UInt16 = None
class aioeos.types.EosPermissionLevelWeight (permission, weight)

    permission: EosPermissionLevel = None
    weight: UInt16 = None
class aioeos.types.EosWaitWeight (wait_sec, weight)

    wait_sec: UInt32 = None
    weight: UInt16 = None
class aioeos.types.EosAuthority (threshold=1, keys=<factory>, accounts=<factory>,
                                waits=<factory>)

    accounts: List[EosPermissionLevelWeight] = None
    keys: List[EosKeyWeight] = None
    threshold: int = 1
    waits: List[EosWaitWeight] = None
class aioeos.types.EosAction (account, name, authorization, data)

    account: Name = None
    authorization: List[EosPermissionLevel] = None
    data: AbiActionPayload = None
    name: Name = None
class aioeos.types.EosTransaction (expiration=<factory>, ref_block_num=0,
                                   ref_block_prefix=0, max_net_usage_words=0,
                                   max_cpu_usage_ms=0, delay_sec=0, context_free_actions=<factory>,
                                   actions=<factory>, transaction_extensions=<factory>)

    actions: List[EosAction] = None
    context_free_actions: List[EosAction] = None
    delay_sec: int = 0
    expiration: TimePointSec = None
    max_cpu_usage_ms: int = 0
    max_net_usage_words: int = 0
```

```
ref_block_num: int = 0
ref_block_prefix: int = 0
transaction_extensions: List[EosExtension] = None
```


CHANGELOG

3.1 1.0.2 (10.04.2020)

- Tested library against RPC node,
- EosAccount API,
- Support using EOS ABI objects as action payloads,
- Signing support in RPC client,
- Cached chain ID,
- Simplified Getting Started guide,
- EOSKey renamed to EosKey

3.2 1.0.1 (03.04.2020)

- Improved docs and “Getting Started” guide,
- Added .coveragerc, fixed abstract class coverage,
- Added mypy,
- Fixed typechecking errors,
- Added docstrings,
- Added missing validation,
- Dropped custom String type,
- Cleaned up serializer.py,
- Bumped base58 to 2.0.0,
- Added tests for RPC client,
- Fixed typehints in docs,
- Added check for building docs

3.3 1.0.0 (01.04.2020)

- Initial release

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

`aioeos.exceptions`, 5
`aioeos.keys`, 6
`aioeos.serializer`, 6
`aioeos.types`, 9

INDEX

A

AbiActionPayloadSerializer (class in *aioeos.serializer*), 6
AbiBytes (in module *aioeos.types*), 9
AbiBytesSerializer (class in *aioeos.serializer*), 6
AbiListSerializer (class in *aioeos.serializer*), 7
AbiNameSerializer (class in *aioeos.serializer*), 7
AbiObjectSerializer (class in *aioeos.serializer*), 7
AbiStringSerializer (class in *aioeos.serializer*), 7
AbiTimePointSecSerializer (class in *aioeos.serializer*), 7
AbiTimePointSerializer (class in *aioeos.serializer*), 8
account (*aioeos.types.EosAction* attribute), 10
accounts (*aioeos.types.EosAuthority* attribute), 10
actions (*aioeos.types.EosTransaction* attribute), 10
actor (*aioeos.types.EosPermissionLevel* attribute), 10
aioeos.exceptions (module), 5
aioeos.keys (module), 6
aioeos.serializer (module), 6
aioeos.types (module), 9
authorization (*aioeos.types.EosAction* attribute), 10

B

BaseAbiObject (class in *aioeos.types*), 9
BaseSerializer (class in *aioeos.serializer*), 8
BasicTypeSerializer (class in *aioeos.serializer*), 8

C

context_free_actions (*aioeos.types.EosTransaction* attribute), 10

D

data (*aioeos.types.EosAction* attribute), 10
delay_sec (*aioeos.types.EosTransaction* attribute), 10
deserialize() (*aioeos.serializer.AbiActionPayloadSerializer* method), 6

deserialize() (*aioeos.serializer.AbiBytesSerializer* method), 6
deserialize() (*aioeos.serializer.AbiListSerializer* method), 7
deserialize() (*aioeos.serializer.AbiNameSerializer* method), 7
deserialize() (*aioeos.serializer.AbiObjectSerializer* method), 7
deserialize() (*aioeos.serializer.AbiStringSerializer* method), 7
deserialize() (*aioeos.serializer.AbiTimePointSecSerializer* method), 7
deserialize() (*aioeos.serializer.AbiTimePointSerializer* method), 8
deserialize() (*aioeos.serializer.BaseSerializer* method), 8
deserialize() (*aioeos.serializer.BasicTypeSerializer* method), 8
deserialize() (*aioeos.serializer.VarUIntSerializer* method), 8
deserialize() (in module *aioeos.serializer*), 8

E

EosAccountDoesntExistException, 5
EosAccountExistsException, 5
EosAction (class in *aioeos.types*), 10
EosActionValidateException, 5
EosAssertMessageException, 5
EosAuthority (class in *aioeos.types*), 10
EosDeadlineException, 5
EosKey (class in *aioeos.keys*), 6
EosKeyWeight (class in *aioeos.types*), 10
EosMissingTaposFieldsException, 5
EosPermissionLevel (class in *aioeos.types*), 9
EosPermissionLevelWeight (class in *aioeos.types*), 10
EosRamUsageExceededException, 5
EosRpcException, 5
EosSerializerAbiNameInvalidCharactersException, 5
EosSerializerAbiNameTooLongException, 5
EosSerializerException, 5

- EosSerializerUnsupportedTypeException, 5
- EosTransaction (class in aioeos.types), 10
- EosTxCpuUsageExceededException, 5
- EosTxNetUsageExceededException, 6
- EosWaitWeight (class in aioeos.types), 10
- expiration (aioeos.types.EosTransaction attribute), 10
- ## F
- Float32 (in module aioeos.types), 9
- Float64 (in module aioeos.types), 9
- ## G
- get_abi_type_serializer() (in module aioeos.serializer), 8
- ## I
- Int16 (in module aioeos.types), 9
- Int32 (in module aioeos.types), 9
- Int64 (in module aioeos.types), 9
- Int8 (in module aioeos.types), 9
- is_abi_object() (in module aioeos.types), 9
- ## K
- key (aioeos.types.EosKeyWeight attribute), 10
- keys (aioeos.types.EosAuthority attribute), 10
- ## M
- max_cpu_usage_ms (aioeos.types.EosTransaction attribute), 10
- max_net_usage_words (aioeos.types.EosTransaction attribute), 10
- ## N
- name (aioeos.types.EosAction attribute), 10
- Name (in module aioeos.types), 9
- ## P
- permission (aioeos.types.EosPermissionLevel attribute), 10
- permission (aioeos.types.EosPermissionLevelWeight attribute), 10
- ## R
- ref_block_num (aioeos.types.EosTransaction attribute), 10
- ref_block_prefix (aioeos.types.EosTransaction attribute), 11
- ## S
- serialize() (aioeos.serializer.AbiActionPayloadSerializer method), 6
- serialize() (aioeos.serializer.AbiBytesSerializer method), 6
- serialize() (aioeos.serializer.AbiListSerializer method), 7
- serialize() (aioeos.serializer.AbiNameSerializer method), 7
- serialize() (aioeos.serializer.AbiObjectSerializer method), 7
- serialize() (aioeos.serializer.AbiStringSerializer method), 7
- serialize() (aioeos.serializer.AbiTimePointSecSerializer method), 8
- serialize() (aioeos.serializer.AbiTimePointSerializer method), 8
- serialize() (aioeos.serializer.BaseSerializer method), 8
- serialize() (aioeos.serializer.BasicTypeSerializer method), 8
- serialize() (aioeos.serializer.VarUIntSerializer method), 8
- serialize() (in module aioeos.serializer), 9
- sign() (aioeos.keys.EosKey method), 6
- ## T
- threshold (aioeos.types.EosAuthority attribute), 10
- TimePoint (in module aioeos.types), 9
- TimePointSec (in module aioeos.types), 9
- to_key_weight() (aioeos.keys.EosKey method), 6
- to_public() (aioeos.keys.EosKey method), 6
- to_pvt() (aioeos.keys.EosKey method), 6
- to_wif() (aioeos.keys.EosKey method), 6
- transaction_extensions (aioeos.types.EosTransaction attribute), 11
- ## U
- UInt16 (in module aioeos.types), 9
- UInt32 (in module aioeos.types), 9
- UInt64 (in module aioeos.types), 9
- UInt8 (in module aioeos.types), 9
- ## V
- VarUInt (in module aioeos.types), 9
- VarUIntSerializer (class in aioeos.serializer), 8
- verify() (aioeos.keys.EosKey method), 6
- ## W
- wait_sec (aioeos.types.EosWaitWeight attribute), 10
- waits (aioeos.types.EosAuthority attribute), 10
- weight (aioeos.types.EosKeyWeight attribute), 10
- weight (aioeos.types.EosPermissionLevelWeight attribute), 10
- weight (aioeos.types.EosWaitWeight attribute), 10