viz-python-lib

Release 0.1.0

Vladimir Kamarzin

Jun 29, 2020

CONTENTS

This library is in alpha state, API unstable

Built on top of python-graphenelib

CHAPTER

ONE

INSTALLATION

Dependencies:

```
sudo apt-get install libffi-dev libssl-dev python-dev
```

Current published version could be installed via

```
pip install viz-python-lib
```

Manual installation:

Install poetry

```
cd viz-python-lib/
poetry install
```

CHAPTER

TWO

BASIC READ QUERY EXAMPLE:

```
from viz import Client
from pprint import pprint

node = 'wss://solox.world/ws'

viz = Client(node=node)
pprint(viz.info())
```

Direct RPC calls:

```
viz.rpc.some_rpc_method()
```

CHAPTER

THREE

CONTENTS

3.1 Version history

We follow Semantic Versions.

3.1.1 Version 0.1.0

- Initial release

3.2 API Reference

This page contains auto-generated API reference documentation[1].

3.2.1 `viz`

Submodules

`viz.account`

Module Contents

`HistoryGenerator`

`class Account` (*account_name: str*, *blockchain_instance: Optional['Client'] = None*)

Bases: `dict`

private-bases This class allows to easily access Account data.

[1] Created with sphinx-autoapi

param str account_name Name of the account

param viz.viz.Client blockchain_instance Client instance

property balances (*self*)
Shortcut to *get_balances()*

property energy (*self*)
Account energy at the moment of last use (stale)

refresh (*self*)
Loads account object from blockchain.

get_balances (*self*)
Obtain account balances.
Returns dict with balances like `{'VIZ': 49400000.0, 'SHARES': 0.0}`

current_energy (*self*)
Returns current account energy (actual data, counts regenerated energy)

virtual_op_count (*self*)
Returns number of virtual ops performed by this account.

get_withdraw_routes (*self*, *type_: str = 'all'*)
Get vesting withdraw routes.
Parameters **type** – route type, one of *all*, *incoming*, *outgoing*
Returns list with routes
Example return:

```
[
    {
        'from_account': 'alice',
        'to_account': 'bob',
        'percent': 10000,
        'auto_vest': False
    }
]
```

get_account_history (*self*, *index: int*, *limit: int*, *start: Optional[int] = None*, *stop: Optional[int] = None*, *order: int = -1*, *filter_by: Optional[Union[str, List[str]]] = None*, *raw_output: bool = False*)
A generator over get_account_history RPC.

It offers serialization, filtering and fine grained iteration control.

Note: This method is mostly for internal use, probably you need *history()*.

Parameters
- **index** (*int*) – start index for get_account_history
- **limit** (*int*) – How many items in account history will be scanned (any ops, not only filtered)
- **start** (*int*) – (Optional) skip items until this index
- **stop** (*int*) – (Optional) stop iteration early at this index
- **order** – (1, -1): 1 for chronological, -1 for reverse order
- **filter_by** (*str, list*) – filter out all but these operations
- **raw_output** (*bool*) – (Defaults to False). If True, return history in steemd format (unchanged).

history(*self*, *filter_by: Optional[Union[str, List[str]]] = None*, *start: int = 0*, *batch_size: int = 1000*, *raw_output: bool = False*, *limit: int = -1*)

Stream account history in chronological order.

This generator yields history items which may be in list or dict form depending on `raw_output`.

Parameters

- **filter_by** (*str, list*) – filter out all but these operations
- **start** (*int*) – (Optional) skip items until this index
- **batch_size** (*int*) – (Optional) request as many items from API in each chunk
- **raw_output** (*bool*) – (Defaults to False). If True, return history in steemd format (unchanged).
- **limit** (*int*) – (Optional) limit number of filtered items to this amount (-1 means unlimited). This is a rough limit, actual results could be a bit longer

Returns number of ops

Non-raw output example of yielded item:

```
{
    'from': 'viz',
    'to': 'null',
    'amount': '1.000 VIZ',
    'memo': 'test',
    'trx_id': '592010ade718c91a81cba3b8378c35ed81d23f23',
    'block': 5,
    'trx_in_block': 0,
    'op_in_trx': 0,
    'virtual_op': 0,
    'timestamp': '2020-05-19T08:10:47',
    'account': 'viz',
    'type': 'transfer',
    '_id': 'd1ed77ae861bb1ecc26a82dd275cc80e5ac124a6',
    'index': 0,
}
```

Raw output example of single item:

```
[
    0,
    {
        'trx_id': '592010ade718c91a81cba3b8378c35ed81d23f23',
        'block': 5,
        'trx_in_block': 0,
        'op_in_trx': 0,
        'virtual_op': 0,
        'timestamp': '2020-05-19T08:10:47',
        'op': ['transfer', {'from': 'viz', 'to': 'null', 'amount': '1.
↪000 VIZ', 'memo': 'test'}],
    },
]
```

history_reverse(*self*, *filter_by: Optional[Union[str, List[str]]] = None*, *batch_size: int = 1000*, *raw_output: bool = False*, *limit: int = -1*)

Stream account history in reverse chronological order.

This generator yields history items which may be in list or dict form depending on `raw_output`. Output is similar to *history()*.

Parameters

- **filter_by** (*str, list*) – filter out all but these operations

- **batch_size** (*int*) – (Optional) request as many items from API in each chunk
- **raw_output** (*bool*) – (Defaults to False). If True, return history in steemd format (unchanged).
- **limit** (*int*) – (Optional) limit number of filtered items to this amount (-1 means unlimited). This is a rough limit, actual results could be a bit longer

Returns number of ops

__contains__()
True if the dictionary has the specified key, else False.

__delattr__()
Implement delattr(self, name).

__delitem__()
Delete self[key].

__dir__()
Default dir() implementation.

__eq__()
Return self==value.

__format__()
Default object formatter.

__ge__()
Return self>=value.

__getattribute__()
Return getattr(self, name).

__getitem__()
x.__getitem__(y) <==> x[y]

__gt__()
Return self>value.

__iter__()
Implement iter(self).

__le__()
Return self<=value.

__len__()
Return len(self).

__lt__()
Return self<value.

__ne__()
Return self!=value.

__reduce__()
Helper for pickle.

__reduce_ex__()
Helper for pickle.

__repr__()
Return repr(self).

__setattr__()
Implement setattr(self, name, value).

__setitem__()
Set self[key] to value.

__sizeof__()
D.__sizeof__() -> size of D in memory, in bytes

__str__()
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

clear()
D.clear() -> None. Remove all items from D.

copy()
D.copy() -> a shallow copy of D

get()
Return the value for key if key is in the dictionary, else default.

items()
D.items() -> a set-like object providing a view on D's items

keys()
D.keys() -> a set-like object providing a view on D's keys

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

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

setdefault()
Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

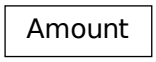
update()
D.update([E,]**F) -> None. Update D from dict/iterable E and F. If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

values()
D.values() -> an object providing a view on D's values

viz.amount

Module Contents

class Amount(*amount_string='0 VIZ'*)
Bases: dict



private-bases This class helps deal and calculate with the different assets on the chain.

param str amount_string Amount string as used by the backend (e.g. "10 VIZ")

__repr__

__truediv__

__truemul__

property amount(*self*)

property symbol(*self*)

property asset(*self*)

__str__(*self*)

__float__(*self*)

__int__(*self*)

__add__(*self*, *other*)

__sub__(*self*, *other*)

__mul__(*self*, *other*)

__floordiv__(*self*, *other*)

__div__(*self*, *other*)

__mod__(*self*, *other*)

__pow__(*self*, *other*)

__iadd__(*self*, *other*)

__isub__(*self*, *other*)

__imul__(*self*, *other*)

__idiv__(*self*, *other*)

__ifloordiv__(*self*, *other*)

__imod__(*self*, *other*)

__ipow__(*self*, *other*)

__lt__ (*self*, *other*)

__le__ (*self*, *other*)

__eq__ (*self*, *other*)

__ne__ (*self*, *other*)

__ge__ (*self*, *other*)

__gt__ (*self*, *other*)

__contains__()
True if the dictionary has the specified key, else False.

__delattr__()
Implement delattr(self, name).

__delitem__()
Delete self[key].

__dir__()
Default dir() implementation.

__format__()
Default object formatter.

__getattribute__()
Return getattr(self, name).

__getitem__()
x.__getitem__(y) <==> x[y]

__iter__()
Implement iter(self).

__len__()
Return len(self).

__reduce__()
Helper for pickle.

__reduce_ex__()
Helper for pickle.

__setattr__()
Implement setattr(self, name, value).

__setitem__()
Set self[key] to value.

__sizeof__()
D.__sizeof__() -> size of D in memory, in bytes

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

clear()
D.clear() -> None. Remove all items from D.

copy()
D.copy() -> a shallow copy of D

get()
Return the value for key if key is in the dictionary, else default.

items()
D.items() -> a set-like object providing a view on D's items

keys()
D.keys() -> a set-like object providing a view on D's keys

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

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

setdefault()
Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

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

values()
D.values() -> an object providing a view on D's values

viz.block

Module Contents

class Block
Bases: graphenecommon.block.Block

private-bases Read a single block from the chain.

param int block block number

param viz.viz.Client blockchain_instance Client instance

param bool lazy Use lazy loading

Instances of this class are dictionaries that come with additional methods (see below) that allow dealing with a block and it's corresponding functions.

```
from viz.block import Block
block = Block(1)
print(block)
```

Note: This class comes with its own caching function to reduce the load on the API server. Instances of this class can be refreshed with `Account.refresh()`.

define_classes(*self*)

class BlockHeader

Bases: graphenecommon.block.BlockHeader

private-bases

define_classes(*self*)

viz.blockchain

Module Contents

class Blockchain

Bases: graphenecommon.blockchain.Blockchain

private-bases This class allows to access the blockchain and read data from it.

param viz.viz.Client blockchain_instance Client instance

param str mode Irreversible block (`irreversible`) or actual head block (`head`) (default: *irreversible*)

param int max_block_wait_repetition maximum wait time for next block is `max_block_wait_repetition * block_interval` (default 3)

This class let's you deal with blockchain related data and methods.

static hash_op(*event: dict*)

This method generates a hash of blockchain operation.

define_classes(*self*)

get_block_interval(*self*)
Override class from graphenelib because our API is different.

stream_from(*self*, *start_block: Optional[int] = None*, *end_block: Optional[int] = None*, *batch_operations: bool = False*, *full_blocks: bool = False*, *only_virtual_ops: bool = False*)
This call yields raw blocks or operations depending on `full_blocks` param.

By default, this generator will yield operations, one by one. You can choose to yield lists of operations, batched to contain all operations for each block with `batch_operations=True`. You can also yield full blocks instead, with `full_blocks=True`.

Parameters

- **start_block** (*int*) – Block to start with. If not provided, current (head) block is used.
- **end_block** (*int*) – Stop iterating at this block. If not provided, this generator will run forever (streaming mode).
- **batch_operations** (*bool*) – (Defaults to False) Rather than yielding operations one by one, yield a list of all operations for each block.
- **full_blocks** (*bool*) – (Defaults to False) Rather than yielding operations, return raw, unedited blocks as provided by blokchain_instance. This mode will NOT include virtual operations.
- **only_virtual_ops** (*bool*) – stream only virtual operations

stream(*self*, *filter_by: Optional[Union[str, List[str]]] = None*, *start_block: Optional[int] = None*, *end_block: Optional[int] = None*, *raw_output: bool = False*)
Yield a stream of specific operations, starting with current head block.

This method can work in 2 modes: 1. Whether only real operations are requested, it will use get_block() API call, so you don't need to have neigher operation_history nor accunt_history plugins enabled. 2. Whether you're requesting any of the virtual operations, your node should have operation_history or accunt_history plugins enabled and appropriate settings for the history-related params should be set (history-start-block, history-whitelist-ops or history-blacklist-ops).

The dict output is formated such that `type` caries the operation type, timestamp and block_num are taken from the block the operation was stored in and the other key depend on the actual operation.

Parameters

- **filter_by** (*str, list*) – List of operations to filter for
- **start_block** (*int*) – Block to start with. If not provided, current (head) block is used.
- **end_block** (*int*) – Stop iterating at this block. If not provided, this generator will run forever (streaming mode).
- **raw_output** (*bool*) – when streaming virtual ops, yield raw ops instead of extended ops format

Example op when streaming virtual ops, `raw_output = False`:

```
{
    '_id': 'e2fabb498706edfccd1114921f05d95e8fd64e4c',
    'type': 'witness_reward',
    'timestamp': '2020-05-29T19:07:48',
    'block_num': 1,
    'trx_id': '0000000000000000000000000000000000000000',
    'witness': 'committee',
    'shares': '0.032999 SHARES',
}
```

Virtual op with `raw_output = True`:

```
{
    'trx_id': '0000000000000000000000000000000000000000',
    'block': 1,
    'trx_in_block': 65535,
    'op_in_trx': 0,
    'virtual_op': 1,
    'timestamp': '2020-05-29T19:28:08',
    'op': ['witness_reward', {'witness': 'committee', 'shares': '0.
↪032999 SHARES'}],
}
```

Real op example:

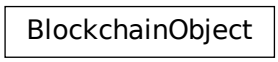
```
{
    'type': 'transfer',
    'timestamp': '2020-05-29T19:20:07',
    'block_num': 6,
    'from': 'viz',
    'to': 'alice',
    'amount': '1.000 VIZ',
    'memo': 'test stream',
}
```

viz.blockchainobject

Module Contents

class BlockchainObject

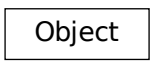
Bases: graphenecommon.blockchainobject.BlockchainObject



private-bases

class Object

Bases: graphenecommon.blockchainobject.Object



private-bases

perform_id_tests = False

viz.converter

Module Contents

class Converter(*blockchain_instance=None*)
Bases: object

Converter simplifies the handling of different metrics of the blockchain.

Parameters **blockchain_instance** (*Steemd*) – Steemd() instance to use when accessing a RPC

core_per_share(*self*)
Obtain CORE_TOKEN/SHARES ratio.

shares_to_core(*self*, *shares*)
Obtain CORE tokens representation of SHARES.

Parameters **shares** (*number*) – SHARES to convert to CORE

core_to_shares(*self*, *amount*)
Obtain SHARES from CORE tokens.

Parameters **amount** (*number*) – amount of CORE tokens to convert

viz.exceptions

Module Contents

exception BaseException
Bases: Exception

private-bases Base excepsion class.

class __cause__
exception cause

class __context__
exception context

class __suppress_context__

class __traceback__

class args

__delattr__()
Implement delattr(self, name).

__dir__()
Default dir() implementation.

__eq__()
Return self==value.

__format__()
Default object formatter.

__ge__()
Return self>=value.

__getattribute__()
Return getattr(self, name).

__gt__()
Return self>value.

__hash__()
Return hash(self).

__le__()
Return self<=value.

__lt__()
Return self<value.

__ne__()
Return self!=value.

__reduce__()

__reduce_ex__()
Helper for pickle.

__repr__()
Return repr(self).

__setattr__()
Implement setattr(self, name, value).

__setstate__()

__sizeof__()
Size of object in memory, in bytes.

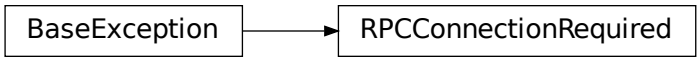
__str__()
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception RPCConnectionRequired
Bases: *viz.exceptions.BaseException*



private-bases An RPC connection is required.

class __cause__
exception cause

class __context__
exception context

class __suppress_context__

class __traceback__

class args

__delattr__()
Implement delattr(self, name).

__dir__()
Default dir() implementation.

__eq__()
Return self==value.

__format__()
Default object formatter.

__ge__()
Return self>=value.

__getattribute__()
Return getattr(self, name).

__gt__()
Return self>value.

__hash__()
Return hash(self).

__le__()
Return self<=value.

__lt__()
Return self<value.

__ne__()
Return self!=value.

__reduce__()

__reduce_ex__()
Helper for pickle.

__repr__()
Return repr(self).

__setattr__()
Implement setattr(self, name, value).

__setstate__()

__sizeof__()
Size of object in memory, in bytes.

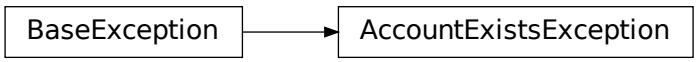
__str__()
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception AccountExistsException
Bases: *viz.exceptions.BaseException*



private-bases The requested account already exists.

class __cause__
exception cause

class __context__
exception context

class __suppress_context__

class __traceback__

class args

__delattr__()
Implement delattr(self, name).

__dir__()
Default dir() implementation.

__eq__()
Return self==value.

__format__()
Default object formatter.

__ge__()
Return self>=value.

__getattribute__()
Return getattr(self, name).

__gt__()
Return self>value.

__hash__()
Return hash(self).

__le__()
Return self<=value.

__lt__()
Return self<value.

__ne__()
Return self!=value.

__reduce__()

__reduce_ex__()
Helper for pickle.

__repr__()
Return repr(self).

__setattr__()
Implement setattr(self, name, value).

__setstate__()

__sizeof__()
Size of object in memory, in bytes.

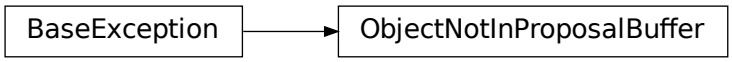
__str__()
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception ObjectNotInProposalBuffer
Bases: *viz.exceptions.BaseException*



private-bases Object was not found in proposal.

class __cause__
exception cause

class __context__
exception context

class __suppress_context__

class __traceback__

class args

__delattr__()
Implement delattr(self, name).

__dir__()
Default dir() implementation.

__eq__()
Return self==value.

__format__()
Default object formatter.

__ge__()
Return self>=value.

__getattribute__()
Return getattr(self, name).

__gt__()
Return self>value.

__hash__()
Return hash(self).

__le__()
Return self<=value.

__lt__()
Return self<value.

__ne__()
Return self!=value.

__reduce__()

__reduce_ex__()
Helper for pickle.

__repr__()
Return repr(self).

__setattr__()
Implement setattr(self, name, value).

__setstate__()

__sizeof__()
Size of object in memory, in bytes.

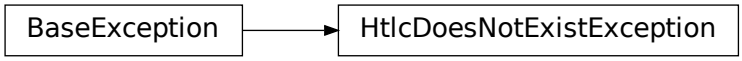
__str__()
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception HtlcDoesNotExistException
Bases: *viz.exceptions.BaseException*



private-bases HTLC object does not exist.

class __cause__
exception cause

class __context__
exception context

class __suppress_context__

class __traceback__

class args

__delattr__()
Implement delattr(self, name).

__dir__()
Default dir() implementation.

__eq__()
Return self==value.

__format__()
Default object formatter.

__ge__()
Return self>=value.

__getattribute__()
Return getattr(self, name).

__gt__()
Return self>value.

__hash__()
Return hash(self).

__le__()
Return self<=value.

__lt__()
Return self<value.

__ne__()
Return self!=value.

__reduce__()

__reduce_ex__()
Helper for pickle.

__repr__()
Return repr(self).

__setattr__()
Implement setattr(self, name, value).

__setstate__()

__sizeof__()
Size of object in memory, in bytes.

__str__()
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

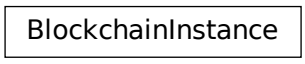
This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

viz.instance

Module Contents

class BlockchainInstance(**args*, ***kwargs*)
Bases: graphenecommon.instance.AbstractBlockchainInstanceProvider



private-bases This is a class that allows compatibility with previous naming conventions.

property viz(*self*)
Alias for the specific blockchain.

get_instance_class(*self*)
Should return the Chain instance class, e.g. *viz.Client*

shared_blockchain_instance()

set_shared_blockchain_instance(*instance*)

set_shared_config(*config*)

shared_chain_instance

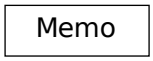
set_shared_chain_instance

viz.memo

Module Contents

class Memo

Bases: graphenecommon.memo.Memo



private-bases Deals with Memos that are attached to a transfer.

param viz.account.Account from_account Account that has sent the memo

param viz.account.Account to_account Account that has received the memo

param viz.viz.Client blockchain_instance Client instance

A memo is encrypted with a shared secret derived from a private key of the sender and a public key of the receiver. Due to the underlying mathematics, the same shared secret can be derived by the private key of the receiver and the public key of the sender. The encrypted message is perturbed by a nonce that is part of the transmitted message.

```
from viz.memo import Memo
m = Memo("alice", "bob")
m.unlock_wallet("secret")
enc = (m.encrypt("foobar"))
print(enc)
print(m.decrypt(enc))
>> foobar
```

To decrypt a memo, simply use

```
from viz.memo import Memo
m = Memo()
m.blockchain.wallet.unlock("secret")
print(memo.decrypt(op_data["memo"]))
```

if `op_data` being the payload of a transfer operation.

define_classes(*self*)

encrypt(*self*, *message: str*)

Encrypt a memo.

This class overriden because upstream assumes memo key is in account['options']['memo_key'], and bitshares memo format is different. We're using Golos memo format.

Parameters **message** (*str*) – clear text memo message
Returns encrypted message
Return type str

decrypt (*self*, *message: str*)
Decrypt a message.
Parameters **message** (*dict*) – encrypted memo message
Returns decrypted message
Return type str

viz.storage

Module Contents

url = wss://ws.viz.ropox.app

appname

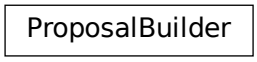
get_default_config_store (**args*, ***kwargs*)

get_default_key_store (*config*, **args*, ***kwargs*)

viz.transactionbuilder

Module Contents

class ProposalBuilder (*author*, *title*, *memo*, *proposal_expiration=None*, *proposal_review=None*, *parent=None*, **args*, ***kwargs*)
Bases: graphenecommon.transactionbuilder.ProposalBuilder



private-bases Proposal Builder allows us to construct an independent Proposal that may later be added to an instance of TransactionBuilder.

param str proposer Account name of the proposing user

param int proposal_expiration Number seconds until the proposal is supposed to expire

param int proposal_review Number of seconds for review of the proposal

param .transactionbuilder.TransactionBuilder Specify your own instance of transaction builder (optional)

param instance blockchain_instance Blockchain instance

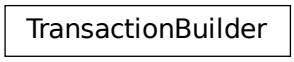
define_classes (*self*)

get_raw (*self*)
Returns an instance of base "Operations" for further processing.

broadcast(*self*)

class TransactionBuilder

Bases: graphenecommon.transactionbuilder.TransactionBuilder



private-bases This class simplifies the creation of transactions by adding operations and signers.

permission_types = ['master', 'active', 'regular']

define_classes(*self*)

add_required_fees(*self*, *ops*, ***kwargs*)
Override this method because steem-like chains doesn't have transaction feed.

appendSigner(*self*, *accounts*, *permission*)
Try to obtain the wif key from the wallet by telling which account and permission is supposed to sign the transaction.

get_block_params(*self*)
Auxiliary method to obtain `ref_block_num` and `ref_block_prefix`.

Requires a websocket connection to a witness node!

viz.utils

Module Contents

json_expand(*json_op*, *key_name='json'*)
Convert a string json object to Python dict in an op.

time_elapsed(*event_time*)
Takes a string time from blockchain event, and returns a time delta from now.

parse_time(*event_time*)
Take a string representation of time from the blockchain, and parse it into datetime object.

time_diff(*time1*, *time2*)

viz.viz

Module Contents

log

class Client

Bases: graphenecommon.chain.AbstractGrapheneChain

Client

private-bases Blockchain network client.

param str node Node to connect to

param str rpcuser RPC user *(optional)*

param str rpcpassword RPC password *(optional)*

param bool nobroadcast Do **not** broadcast a transaction! *(optional)*

param bool debug Enable Debugging *(optional)*

param array,dict,string keys Predefine the wif keys to shortcut the wallet database *(optional)*

param bool offline Boolean to prevent connecting to network (defaults to `False`) *(optional)*

param int expiration Delay in seconds until transactions are supposed to expire *(optional)*

param bool blocking Wait for broadcasted transactions to be included in a block and return full transaction. Blocking is checked inside `broadcast()` *(Default: False)*, *(optional)*

param bool bundle Do not broadcast transactions right away, but allow to bundle operations *(optional)*

Three wallet operation modes are possible:

- **Wallet Database**: Here, the libs load the keys from the locally stored wallet SQLite database (see `storage.py`). To use this mode, simply call `Client()` without the `keys` parameter
- **Providing Keys**: Here, you can provide the keys for your accounts manually. All you need to do is add the wif keys for the accounts you want to use as a simple array using the `keys` parameter to `Client()`.
- **Force keys**: This more is for advanced users and requires that you know what you are doing. Here, the `keys` parameter is a dictionary that overwrite the `active`, `master`, or `memo` keys for any account. This mode is only used for *foreign* signatures!

The purpose of this class it to simplify interaction with blockchain by providing high-level methods instead of forcing user to use RPC methods directly.

The idea is to have a class that allows to do this:

```
from viz import Client
viz = Client()
print(viz.info())
```

define_classes(*self*)

new_proposal(*self*, *title: str*, *memo: str = ''*, *parent: Optional[TransactionBuilder] = None*, *expiration_time: int = 2 * 24 * 60 * 60*, *review_period_time: Optional[int] = None*, *account: str = None*, ***kwargs: Any*)

Create a new proposal.

Proposal is a way to propose some transaction to another account. Primary usecase is a multisig account which requires several members approval to perform an operation.

Parameters

- **title** (*str*) – title of proposed transaction
- **memo** (*str*) – may be a description of the proposal
- **parent** (TransactionBuilder) – TransactionBuilder instance to add proposal to
- **expiration_time** (*int*) – maximum time allowed for transaction
- **review_period_time** (*int*) – time to make a decision of the transaction participants
- **account** (*str*) – author of proposed transaction

Example usage:

```
from viz import Client
viz = Client()
proposal = viz.new_proposal('title', 'test proposal', account='alice')
viz.transfer("null", 1, "VIZ", memo="test transfer proposal",␣
↪account=default_account, append_to=proposal)
proposal.broadcast()
```

proposal_update(*self*, *author: str*, *title: str*, *approver: Union[str, list] = None*, *keys: Union[str, list] = None*, *permission: str = 'regular'*, *approve: bool = True*, *account: str = None*)

Update proposal (approve or disapprove)

Parameters

- **author** (*str*) – author of proposed transaction
- **title** (*str*) – title of proposed transaction
- **list approver** (*str,*) – account(s) for approvals, default is account field
- **keys** (*str*) – public key(s) used for multisig accounts (key approval)
- **permission** (*str*) – the required permission type for signing
- **approve** (*str*) – True = approve, False = disapprove
- **account** (*str*) – the account that authorizes the operation

transfer(*self*, *to: str*, *amount: float*, *asset: str*, *memo: str = ''*, *account: Optional[str] = None*, ***kwargs: Any*)

Transfer an asset to another account.

Parameters

- **to** (*str*) – Recipient
- **amount** (*float*) – Amount to transfer
- **asset** (*str*) – Asset to transfer
- **memo** (*str*) – (optional) Memo, may begin with # for encrypted messaging
- **account** (*str*) – (optional) the source account for the transfer if not `default_account`

award(*self*, *receiver: str*, *energy: float*, *memo: str = ''*, *beneficiaries: Optional[List[Dict[str, Union[str, int]]]] = None*, *account: str = None*, ***kwargs: Any*)

Award someone.

Parameters

- **receiver** (*str*) – account name of award receiver
- **energy** (*float*) – energy as 0-100%
- **memo** (*str*) – optional comment
- **beneficiaries** (*list*) – list of dicts, example `[{'account': 'vvk',`

```
'weight':  50}]
```

- **account** (*str*) – initiator account name

custom(*self, id_: str, json: Union[Dict, List], required_active_auths: Optional[List[str]] = None, required_regular_auths: Optional[List[str]] = None*)

Create a custom operation.

Parameters

- **id** (*str*) – identifier for the custom (max length 32 bytes)
- **json** (*dict, list*) – the json data to put into the custom operation
- **required_active_auths** (*list*) – (optional) require signatures from these active auths to make this operation valid
- **required_regular_auths** (*list*) – (optional) require signatures from these regular auths

withdraw_vesting(*self*, *amount: float*, *account: str = None*)

Withdraw SHARES from the vesting account.

Parameters

- **amount** (*float*) – number of SHARES to withdraw over a period
- **account** (*str*) – (optional) the source account for the transfer if not `default_account`

transfer_to_vesting(*self*, *amount: float*, *to: str = None*, *account: str = None*)

Vest free VIZ into vesting.

Parameters

- **amount** (*float*) – number of VIZ to vest
- **to** (*str*) – (optional) the source account for the transfer if not `default_account`
- **account** (*str*) – (optional) the source account for the transfer if not `default_account`

set_withdraw_vesting_route(*self*, *to: str*, *percentage: float = 100*, *account: str = None*, *auto_vest: bool = False*)

Set up a vesting withdraw route. When vesting shares are withdrawn, they will be routed to these accounts based on the specified weights.

To obtain existing withdraw routes, use *get_withdraw_vesting_routes()*

```
a = Account('vvk', blockchain_instance=viz)
pprint(a.get_withdraw_routes())
```

Parameters

- **to** (*str*) – Recipient of the vesting withdrawal
- **percentage** (*float*) – The percent of the withdraw to go to the 'to' account.
- **account** (*str*) – (optional) the vesting account
- **auto_vest** (*bool*) – Set to true if the from account should receive the SHARES as SHARES, or false if it should receive them as CORE. (defaults to `False`)

get_withdraw_vesting_routes(*self*, *account: str*, ***kwargs: str*)

Get vesting withdraw route for an account.

This is a shortcut for `viz.account.Account.get_withdraw_routes()`.

Parameters **account** (*str*) – account name

Returns list with routes

create_account(*self*, *account_name: str*, *json_meta: Optional[Dict[str, Any]] = None*, *password: str = None*, *master_key: str = None*, *active_key: str = None*, *regular_key: str = None*, *memo_key: str = None*, *additional_master_keys: Optional[List[str]] = None*, *additional_active_keys: Optional[List[str]] = None*, *additional_regular_keys: Optional[List[str]] = None*, *additional_master_accounts: Optional[List[str]] = None*, *additional_active_accounts: Optional[List[str]] = None*, *additional_regular_accounts: Optional[List[str]] = None*, *store_keys: bool = True*, *fee: float = 0*, *delegation: float = 0*, *creator: str = None*, *referrer: str = ''*)

Create new account.

The brainkey/password can be used to recover all generated keys (see *vizbase.account* for more details.

By default, this call will use `default_account` to register a new name `account_name` with all keys being derived from a new brain key that will be returned. The corresponding keys will automatically be installed in the wallet.

Note:
Account creations cost a fee that is defined by the network. If you create an account, you will need to pay for that fee!
You can partially pay that fee by delegating SHARES.

> **Warning:** Don't call this method unless you know what you are doing! Be sure to understand what this method does and where to find the private keys for your account.

Note: Please note that this imports private keys (if password is present) into the wallet by default.

Parameters

- **account_name** (*str*) – (**required**) new account name
- **json_meta** (*dict*) – Optional meta data for the account
- **password** (*str*) – Alternatively to providing keys, one can provide a password from which the keys will be derived
- **master_key** (*str*) – Main master key
- **active_key** (*str*) – Main active key
- **regular_key** (*str*) – Main regular key
- **memo_key** (*str*) – Main memo_key
- **additional_master_keys** (*list*) – Additional master public keys
- **additional_active_keys** (*list*) – Additional active public keys
- **additional_regular_keys** (*list*) – Additional regular public keys
- **additional_master_accounts** (*list*) – Additional master account names
- **additional_active_accounts** (*list*) – Additional active account names
- **additional_regular_accounts** (*list*) – Additional regular account names
- **store_keys** (*bool*) – Store new keys in the wallet (default: `True`)

- **fee** (*float*) – (Optional) If set, *creator* pay a fee of this amount, and delegate the rest with SHARES (calculated automatically).
- **delegation** (*float*) – amount of SHARES to be delegated to a new account
- **creator** (*str*) – which account should pay the registration fee (defaults to `default_account`)
- **referrer** (*str*) – account who will be set as referrer, defaults to creator

Raises ***AccountExistsException*** – if the account already exists on the blockchain

_store_keys (*self*, **args*)
Store private keys to local storage.

viz.wallet

Module Contents

class Wallet
Bases: graphenecommon.wallet.Wallet

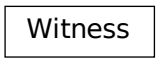
private-bases

define_classes (*self*)

viz.witness

Module Contents

class Witness
Bases: graphenecommon.witness.Witness



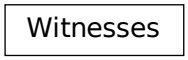
private-bases Read data about a witness in the chain.

param str account_name Name of the witness

param viz blockchain_instance Client() instance to use when accesing a RPC

define_classes (*self*)

class Witnesses
Bases: graphenecommon.witness.Witnesses



private-bases Obtain a list of **active** witnesses and the current schedule.

param bool only_active (False) Only return witnesses that are actively producing blocks

param viz blockchain_instance Client() instance to use when accesing a RPC

define_classes (*self*)

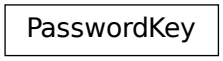
3.2.2 vizbase

Submodules

vizbase.account

Module Contents

class PasswordKey
Bases: graphenebase.account.PasswordKey

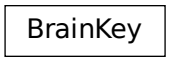


private-bases This class derives a private key given the account name, the role and a password.

It leverages the technology of Brainkeys and allows people to have a secure private key by providing a passphrase only.

prefix

class BrainKey
Bases: graphenebase.account.BrainKey



private-bases Brainkey implementation similar to the graphene-ui web-wallet.

param str brainkey Brain Key

param int sequence Sequence number for consecutive keys

Keys in Graphene are derived from a seed brain key which is a string of 16 words out of a predefined dictionary with 49744 words. It is a simple single-chain key derivation scheme that is not compatible with BIP44 but easy to use.

Given the brain key, a private key is derived as:

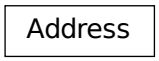
```
privkey = SHA256(SHA512(brainkey + " " + sequence))
```

Incrementing the sequence number yields a new key that can be regenerated given the brain key.

prefix

class Address

Bases: graphenebase.account.Address



private-bases Address class.

This class serves as an address representation for Public Keys.

param str address Base58 encoded address (defaults to None)

param str pubkey Base58 encoded pubkey (defaults to None)

param str prefix Network prefix (defaults to BTS)

Example:

```
Address("BTSFN9r6VYzBK8EKtMewfNbfiGCr56pHDBFi")
```

prefix

class PublicKey

Bases: graphenebase.account.PublicKey

private-bases This class deals with Public Keys and inherits `Address`.

param str pk Base58 encoded public key

param str prefix Network prefix (defaults to `BTS`)

Example::

```
PublicKey("BTS6UtYWWs3rkZGV8JA86qrgkG6tyFksgECefKE1MiH4HkLD8PFGL")
```

Note: By default, graphene-based networks deal with **compressed** public keys. If an **uncompressed** key is required, the method `unCompressed` can be used:

```
PublicKey("xxxxx").unCompressed()
```

---

prefix

class PrivateKey

Bases: `graphenebase.account.PrivateKey`

private-bases Derives the compressed and uncompressed public keys and constructs two instances of `PublicKey`:

param str wif Base58check-encoded wif key

param str prefix Network prefix (defaults to `BTS`)

Example::

```
PrivateKey("5HqUkGuo62BfcJU5vNhTXKJRXuUi9QSE6jp8C3uBJ2BVHtB8WSd")
```

Compressed vs. Uncompressed:

- **`PrivateKey("w-i-f").pubkey`:** Instance of `PublicKey` using compressed key.
- **`PrivateKey("w-i-f").pubkey.address`:** Instance of `Address` using compressed key.
- **`PrivateKey("w-i-f").uncompressed`:** Instance of `PublicKey` using uncompressed key.

- **PrivateKey("w-i-f").uncompressed.address:** Instance of `Address` using uncompressed key.

prefix

vizbase.chains

Module Contents

DEFAULT_PREFIX = VIZ

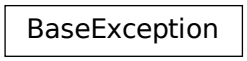
KNOWN_CHAINS

PRECISIONS

vizbase.exceptions

Module Contents

exception BaseException
Bases: `Exception`



private-bases

class __cause__
exception cause

class __context__
exception context

class __suppress_context__

class __traceback__

class args

__delattr__()
Implement delattr(self, name).

__dir__()
Default dir() implementation.

__eq__()
Return self==value.

__format__()
Default object formatter.

__ge__()
Return self>=value.

__getattribute__()
Return getattr(self, name).

__gt__()
Return self>value.

__hash__()
Return hash(self).

__le__()
Return self<=value.

__lt__()
Return self<value.

__ne__()
Return self!=value.

__reduce__()

__reduce_ex__()
Helper for pickle.

__repr__()
Return repr(self).

__setattr__()
Implement setattr(self, name, value).

__setstate__()

__sizeof__()
Size of object in memory, in bytes.

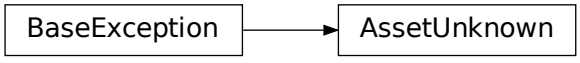
__str__()
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception AssetUnknown
Bases: *vizbase.exceptions.BaseException*



private-bases

class __cause__
exception cause

class __context__
exception context

class __suppress_context__

class __traceback__

class args

__delattr__()
Implement delattr(self, name).

__dir__()
Default dir() implementation.

__eq__()
Return self==value.

__format__()
Default object formatter.

__ge__()
Return self>=value.

__getattribute__()
Return getattr(self, name).

__gt__()
Return self>value.

__hash__()
Return hash(self).

__le__()
Return self<=value.

__lt__()
Return self<value.

__ne__()
Return self!=value.

__reduce__()

__reduce_ex__()
Helper for pickle.

__repr__()
Return repr(self).

__setattr__()
Implement setattr(self, name, value).

__setstate__()

__sizeof__()
Size of object in memory, in bytes.

__str__()
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

vizbase.memo

Module Contents

init_aes(*shared_secret*, *nonce*)
Initialize AES instance.

Parameters

- **shared_secret** (*hex*) – Shared Secret to use as encryption key
- **nonce** (*int*) – Random nonce

Returns AES instance and checksum of the encryption key

Return type length 2 tuple

encode_memo(*priv*, *pub*, *nonce*, *message*, ***kwargs*)
Encode a message with a shared secret between Alice and Bob.

Parameters

- **priv** (PrivateKey) – Private Key (of Alice)
- **pub** (PublicKey) – Public Key (of Bob)
- **nonce** (*int*) – Random nonce
- **message** (*str*) – Memo message

Returns Encrypted message

Return type hex

decode_memo(*priv*, *message*)
Decode a message with a shared secret between Alice and Bob.

Parameters

- **priv** (PrivateKey) – Private Key (of Bob)
- **message** (*base58encoded*) – Encrypted Memo message

Returns Decrypted message

Return type str

Raises **ValueError** – if message cannot be decoded as valid UTF-8 string

involved_keys(*message*)
decode structure.

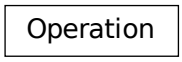
_pad(*raw_message*, *bs*)

_unpad(*raw_message*, *bs*)

vizbase.objects

Module Contents

class Operation
Bases: graphenebase.objects.Operation



private-bases Need to overwrite a few attributes to load proper operations from viz.

module = vizbase.operations

operations

class Amount(*d*)

__bytes__(*self*)

__str__(*self*)

class Beneficiary(**args*, ***kwargs*)
Bases: graphenebase.objects.GrapheneObject

private-bases

class Memo(**args*, ***kwargs*)
Bases: graphenebase.objects.GrapheneObject

private-bases

class Permission(**args*, ***kwargs*)
Bases: graphenebase.objects.GrapheneObject

private-bases

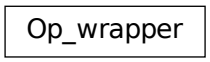
class ChainPropertiesVariant(*props*)
Bases: graphenebase.types.Static_variant

private-bases

class ChainProperties(**args*, ***kwargs*)
Bases: graphenebase.objects.GrapheneObject

private-bases

class Op_wrapper(**args*, ***kwargs*)
Bases: graphenebase.objects.GrapheneObject



private-bases

vizbase.operationids

Module Contents

OPS = ['vote', 'content', 'transfer', 'transfer_to_vesting', 'withdraw_vesting', 'account_u

operations

VIRTUAL_OPS = ['author_reward', 'curation_reward', 'content_reward', 'fill_vesting_withdraw

vizbase.operations

Module Contents

class Account_create(**args*, ***kwargs*)

Bases: vizbase.objects.GrapheneObject

Account_create

private-bases

class Account_update(**args*, ***kwargs*)

Bases: vizbase.objects.GrapheneObject

Account_update

private-bases

class Account_metadata(**args*, ***kwargs*)

Bases: vizbase.objects.GrapheneObject

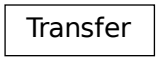
Account_metadata

private-bases

class Award(**args*, ***kwargs*)
Bases: vizbase.objects.GrapheneObject

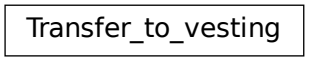
private-bases

class Transfer(**args*, ***kwargs*)
Bases: vizbase.objects.GrapheneObject



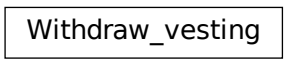
private-bases

class Transfer_to_vesting(**args*, ***kwargs*)
Bases: vizbase.objects.GrapheneObject



private-bases

class Withdraw_vesting(**args*, ***kwargs*)
Bases: vizbase.objects.GrapheneObject

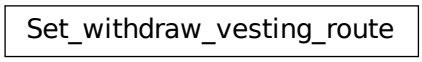


private-bases

class Delegate_vesting_shares(**args*, ***kwargs*)
Bases: vizbase.objects.GrapheneObject

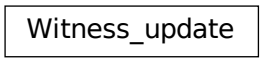
private-bases

class Set_withdraw_vesting_route(**args*, ***kwargs*)
Bases: vizbase.objects.GrapheneObject



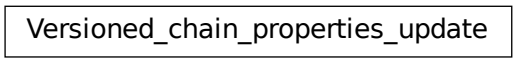
private-bases

class Witness_update(**args*, ***kwargs*)
Bases: vizbase.objects.GrapheneObject



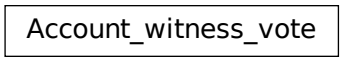
private-bases

class Versioned_chain_properties_update(**args*, ***kwargs*)
Bases: vizbase.objects.GrapheneObject



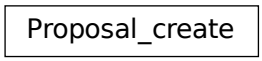
private-bases

class Account_witness_vote(**args*, ***kwargs*)
Bases: vizbase.objects.GrapheneObject



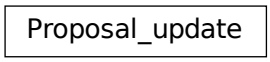
private-bases

class Proposal_create(**args*, ***kwargs*)
Bases: vizbase.objects.GrapheneObject



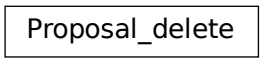
private-bases See libraries/protocol/include/graphene/protocol/proposal_operations.hpp.

class Proposal_update(**args*, ***kwargs*)
Bases: vizbase.objects.GrapheneObject



private-bases

class Proposal_delete(**args*, ***kwargs*)
Bases: vizbase.objects.GrapheneObject



private-bases

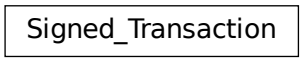
class Custom(**args*, ***kwargs*)
Bases: vizbase.objects.GrapheneObject

private-bases

vizbase.signedtransactions

Module Contents

class Signed_Transaction
Bases: graphenebase.signedtransactions.Signed_Transaction



private-bases Create a signed transaction and offer method to create the signature.

param num refNum parameter ref_block_num (see `getBlockParams`)

param num refPrefix parameter ref_block_prefix (see `getBlockParams`)

param str expiration expiration date

param Array operations array of operations

known_chains

default_prefix

operation_klass

3.2.3 vizapi

Submodules

vizapi.consts

Module Contents

API

vizapi.exceptions

Module Contents

decode_rpc_error_msg(*exc: Exception*) → str
Helper function to decode the raised Exception and give it a python Exception class.

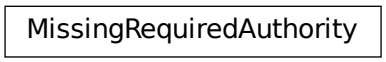
Exception text usually consists of two lines, in raw:

`Assert Exception (10)\namount.amount > 0:  Cannot transfer a negative amount (aka:  stealing)\n\n` or

`missing required active authority (3010000)\nMissing Active Authority ["viz"]\n\n\n`

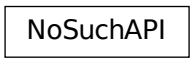
We're omitting the fist line and returning meaningful second line, stripping trailing newlines.

class MissingRequiredAuthority
Bases: grapheneapi.exceptions.RPCError



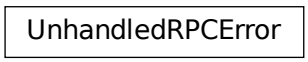
private-bases

class NoSuchAPI
Bases: grapheneapi.exceptions.RPCError



private-bases

class UnhandledRPCError
Bases: grapheneapi.exceptions.RPCError

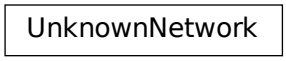


private-bases

class ReadLockFail
Bases: grapheneapi.exceptions.RPCError

private-bases

class UnknownNetwork
Bases: grapheneapi.exceptions.RPCError



private-bases

vizapi.noderpc

Module Contents

log

class NodeRPC(**args*, ***kwargs*)
Bases: grapheneapi.api.Api

private-bases Redefine graphene Api class.

Class wraps communications with API nodes via proxying requests to lower-level *Rpc* class and it's implementations *Websocket* and *Http*.

To enable RPC debugging:

```
log = logging.getLogger('vizapi')
log.setLevel(logging.DEBUG)
log.addHandler(logging.StreamHandler())
```

post_process_exception(*self*, *error: Exception*)
Process error response and raise proper exception.

Called from *__getattr__()*, which catches RPCError exception which raised by `Rpc.parse_response()` in Rpc class.

Parameters **error** – exception

updated_connection(*self*)

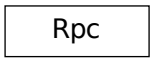
get_network(*self*)
Cache connected network info.

This avoids multiple calls of self.get_config()

_get_network(*self*)
Identify the connected network.

This call returns a dictionary with keys chain_id, core_symbol and prefix

class Rpc(**args*, ***kwargs*)
Bases: `grapheneapi.rpc.Rpc`

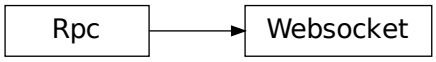


private-bases This class is responsible for making RPC queries.

Original graphene chains (like Bitshares) uses api_id in "params", while Golos and VIZ uses api name here.

__getattr__(*self*, *name*)
Map all methods to RPC calls and pass through the arguments.

class Websocket(**args*, ***kwargs*)
Bases: `grapheneapi.websocket.Websocket`, *vizapi.noderpc.Rpc*



private-bases Interface to API node websocket endpoint.

We have to override Websocket class because we need it to inherit from our own Rpc class.

__getattr__(*self*, *name*)
Map all methods to RPC calls and pass through the arguments.

class Http(**args*, ***kwargs*)
Bases: `grapheneapi.http.Http`, *vizapi.noderpc.Rpc*

private-bases Interface to API node http endpoint.

We have to override Websocket class because we need it to inherit from our own Rpc class.

__getattr__(*self*, *name*)

Map all methods to RPC calls and pass through the arguments.

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

V

INDEX

Symbols

A

B

C

R

S

T

U

V

W