
Aergo Herapy Documentation

Release 1.2.7

aergo

Mar 09, 2020

Contents:

1	What is Herapy?	3
2	Getting Started	5
2.1	Installation	5
2.2	Connecting to Aergo	5
2.3	Creating a new Account	6
2.4	Exporting/Importing an Account	6
2.5	Creating a Transaction	6
2.6	Deploying and calling smart contracts	7
3	Modules	9
3.1	aergo.herapy package	9
3.1.1	Subpackages	9
3.1.2	Submodules	28
3.1.3	aergo.herapy.account module	28
3.1.4	aergo.herapy.aergo module	28
3.1.5	aergo.herapy.comm module	32
3.1.6	aergo.herapy.constants module	33
3.1.7	Module contents	33
3.2	Examples and projects using Herapy	33
4	Indices and tables	35
	Python Module Index	37
	Index	39

Aergo HeraPy is the Python SDK for communicating and interacting with the Aergo blockchain.

CHAPTER 1

What is Herapy?

Herapy is **Aergo**'s Python3 SDK for connecting to Aergo networks.

It contains the following features:

- connect to an **Aergo** blockchain node
- create a new account
- manage an account
- create a transaction
- sign a transaction
- send a transaction
- deploy a smart contract
- call a smart contract
- query a smart contract
- request and verify state Merkle proofs
- and so on.

CHAPTER 2

Getting Started

Let's find out how to use [HeraPy](#) quickly with a few examples.

2.1 Installation

- Python3 (≥ 3.7)

Setup your environment and install aergo-herapy

```
$ cd my_new_project
$ virtualenv -p python3 venv
$ source venv/bin/activate
$ pip install aergo-herapy
```

2.2 Connecting to Aergo

Connecting to Aergo can be done with a public api like 'testnet-api.aergo.io:7845' or by *running your own Aergo node*.

```
1 import aergo.herapy as herapy
2
3 aergo = herapy.Aergo()
4 aergo.connect('testnet-api.aergo.io:7845')
5
6 print(aergo.get_chain_info())
7
8 aergo.disconnect()
```

2.3 Creating a new Account

Connecting to Aergo is optional when creating new accounts with the parameter `skip_state=True`.

```
1 import aergo.herapy as herapy
2
3 aergo = herapy.Aergo()
4
5 # connect to a node to retrieve the account state (nonce, balance...)
6 # aergo.connect('testnet-api.aergo.io:7845')
7 # aergo.new_account()
8
9 # create a new account offline
10 aergo.new_account(skip_state=True)
11
12 # print the address
13 print(aergo.get_address())
14
15 # print the address as bytes
16 print(bytes(aergo.get_address()))
```

2.4 Exporting/Importing an Account

For using an account created in various other SDKs and Aergocli, the preferred method is to import an Aergo encrypted keystore file.

Connecting to a node is optional.

```
1 import aergo.herapy as herapy
2
3 aergo = herapy.Aergo()
4 aergo.new_account(skip_state=True)
5 exp_account = aergo.export_account_to_keystore("keep-safe")
6
7 aergo2 = herapy.Aergo()
8 aergo2.import_account_from_keystore(exp_account, "keep-safe", skip_state=True)
```

2.5 Creating a Transaction

```
1 import aergo.herapy as herapy
2
3 # connect to a node
4 aergo = herapy.Aergo()
5 aergo.connect('testnet-api.aergo.io:7845')
6
7 keystore_file_path = "./my/keystore.json"
8
9 # import account from keystore file and get current nonce
10 aergo.import_account_from_keystore_file(keystore_file_path, "keep-safe")
11
12 # transfer 1 aergo
13 tx, status = aergo.transfer(to_address, 1 * 10**18)
```

(continues on next page)

(continued from previous page)

```
14
15 assert result.status == herapy.CommitStatus.TX_OK
16
17 receipt = aergo.wait_tx_result(tx.tx_hash)
18
19 assert receipt.status == herapy.TxResultStatus.SUCCESS:
```

2.6 Deploying and calling smart contracts

```
1 import aergo.herapy as herapy
2
3 # connect to a node
4 aergo = herapy.Aergo()
5 aergo.connect('testnet-api.aergo.io:7845')
6
7 keystore_file_path = "./my/keystore.json"
8
9 # import account from keystore file and get current nonce
10 aergo.import_account_from_keystore_file(keystore_file_path, "keep-safe")
11
12
13 # deploy a new contract
14 payload = "Compiled contract string"
15 tx, result = aergo.deploy_sc(amount=0, payload=payload, args=1234)
16 assert result.status == herapy.CommitStatus.TX_OK
17
18 receipt = aergo.wait_tx_result(tx.tx_hash)
19 assert receipt.status == herapy.TxResultStatus.CREATED:
20
21 # get address of newly deployed contract
22 sc_address = receipt.contract_address
23
24 # send a transaction to a contract (write)
25 tx, result = aergo.call_sc(sc_address, "lua function name")
26 assert result.status == herapy.CommitStatus.TX_OK
27
28 assert receipt.status == herapy.TxResultStatus.SUCCESS:
29 receipt = aergo.wait_tx_result(tx.tx_hash)
30
31
32 # query a contract function (read-only)
33 return_value = aergo.query_sc(sc_address, "lua function name")
```


The documentation is being written so for other tools and advanced features, please refer to the **herapy.aergo** and **herapy.account** modules description below.

3.1 aergo.herapy package

3.1.1 Subpackages

aergo.herapy.errors package

Submodules

aergo.herapy.errors.InsufficientBalanceError module

exception aergo.herapy.errors.InsufficientBalanceError.**InsufficientBalanceError**(*message*)
Bases: `Exception`

aergo.herapy.errors.conversion_exception module

exception aergo.herapy.errors.conversion_exception.**ConversionException**(*msg*)
Bases: `aergo.herapy.errors.exception.AergoException`

aergo.herapy.errors.exception module

exception aergo.herapy.errors.exception.**AergoException**(*error, exception_type*)
Bases: `Exception`

`Comm = 'Communication Exception'`

```
Conv = 'Conversion Exception'
General = 'General Exception'
exception aergo.herapy.errors.exception.CommunicationException(error)
    Bases: aergo.herapy.errors.exception.AergoException
```

aergo.herapy.errors.general_exception module

```
exception aergo.herapy.errors.general_exception.GeneralException(msg)
    Bases: aergo.herapy.errors.exception.AergoException
```

Module contents

aergo.herapy.grpc package

Submodules

aergo.herapy.grpc.account_pb2 module

aergo.herapy.grpc.account_pb2_grpc module

aergo.herapy.grpc.blockchain_pb2 module

aergo.herapy.grpc.blockchain_pb2_grpc module

aergo.herapy.grpc.metric_pb2 module

aergo.herapy.grpc.metric_pb2_grpc module

aergo.herapy.grpc.node_pb2 module

aergo.herapy.grpc.node_pb2_grpc module

aergo.herapy.grpc.p2p_pb2 module

aergo.herapy.grpc.p2p_pb2_grpc module

aergo.herapy.grpc.pmap_pb2 module

aergo.herapy.grpc.pmap_pb2_grpc module

aergo.herapy.grpc.polarrpc_pb2 module

aergo.herapy.grpc.polarrpc_pb2_grpc module

```
class aergo.herapy.grpc.polarrpc_pb2_grpc.PolarisRPCServiceServicer
    Bases: object
```

AddBLEntry (*request, context*)

BlackList (*request, context*)

CurrentList (*request, context*)

ListBLEntries (*request, context*)

Metric (*request, context*)

Returns node metrics according to request

NodeState (*request, context*)

Returns the current state of this node

RemoveBLEntry (*request, context*)

WhiteList (*request, context*)

class aergo.herapy.grpc.polarrpc_pb2_grpc.**PolarisRPCServiceStub** (*channel*)

Bases: `object`

aergo.herapy.grpc.polarrpc_pb2_grpc.**add_PolarisRPCServiceServicer_to_server** (*servicer, server*)

aergo.herapy.grpc.raft_pb2 module

aergo.herapy.grpc.raft_pb2_grpc module

aergo.herapy.grpc.rpc_pb2 module

aergo.herapy.grpc.rpc_pb2_grpc module

class aergo.herapy.grpc.rpc_pb2_grpc.**AergoRPCServiceServicer**

Bases: `object`

•

AergoRPCService is the main RPC service providing endpoints to interact with the node and blockchain. If not otherwise noted, methods are unary requests.

Blockchain (*request, context*)

Returns current blockchain status (best block's height and hash)

ChainStat (*request, context*)

Returns current chain statistics

ChangeMembership (*request, context*)

Add & remove member of raft cluster

CommitTX (*request, context*)

Commit a signed transaction

CreateAccount (*request, context*)

Create a new account in this node

ExportAccount (*request, context*)

Export account stored in this node

GetABI (*request, context*)

Return ABI stored at contract address

GetAccountVotes (*request, context*)

Return staking, voting info for account

GetAccounts (*request, context*)

Return list of accounts in this node

GetBlock (*request, context*)

Return a single block incl. header and body, queried by hash or number

GetBlockBody (*request, context*)

Return a single block's body, queried by hash or number and list parameters

GetBlockMetadata (*request, context*)

Return a single block's metadata (hash, header, and number of transactions), queried by hash or number

GetBlockTX (*request, context*)

Return information about transaction in block, queried by transaction hash

GetChainInfo (*request, context*)

Returns current blockchain's basic information

GetConfChangeProgress (*request, context*)

Return a status of changeCluster enterprise tx, queried by requestID

GetConsensusInfo (*request, context*)

Returns status of consensus and bps

GetEnterpriseConfig (*request, context*)

Returns enterprise config

GetNameInfo (*request, context*)

Return name information

GetPeers (*request, context*)

Return list of peers of this node and their state

GetReceipt (*request, context*)

Return transaction receipt, queried by transaction hash

GetServerInfo (*request, context*)

Returns configs and statuses of server

GetStaking (*request, context*)

Return staking information

GetState (*request, context*)

Return state of account

GetStateAndProof (*request, context*)

Return state of account, including merkle proof

GetTX (*request, context*)

Return a single transaction, queried by transaction hash

GetVotes (*request, context*)

Return result of vote

ImportAccount (*request, context*)

Import account to this node

ListBlockHeaders (*request, context*)

Returns list of Blocks without body according to request

ListBlockMetadata (*request, context*)

Returns list of block metadata (hash, header, and number of transactions) according to request

ListBlockMetadataStream (*request, context*)

Returns a stream of new block's metadata as they get added to the blockchain

ListBlockStream (*request, context*)

Returns a stream of new blocks as they get added to the blockchain

ListEventStream (*request, context*)

Returns a stream of event as they get added to the blockchain

ListEvents (*request, context*)

Returns list of event

LockAccount (*request, context*)

Lock account in this node

Metric (*request, context*)

Returns node metrics according to request

NodeState (*request, context*)

Returns the current state of this node

QueryContract (*request, context*)

Query a contract method

QueryContractState (*request, context*)

Query contract state

SendTX (*request, context*)

Sign and send a transaction from an unlocked account

SignTX (*request, context*)

Sign transaction with unlocked account

UnlockAccount (*request, context*)

Unlock account in this node

VerifyTX (*request, context*)

Verify validity of transaction

class aergo.herapy.grpc.rpc_pb2_grpc.**AergoRPCServiceStub** (*channel*)

Bases: `object`

•

AergoRPCService is the main RPC service providing endpoints to interact with the node and blockchain. If not otherwise noted, methods are unary requests.

aergo.herapy.grpc.rpc_pb2_grpc.**add_AergoRPCServiceServicer_to_server** (*servicer, server*)

Module contents

grpc package for herapy.

aergo.herapy.obj package

Submodules

aergo.herapy.obj.abi module

```
class aergo.herapy.obj.abi.Abi (abi)
    Bases: object
    Abi stores a contract abi.

    functions

    json ()

    language

    state_variables

    version
```

aergo.herapy.obj.address module

```
class aergo.herapy.obj.address.Address (pubkey: Union[str, bytes, ecdsa.ecdsa.Public_key],
                                         empty: bool = False, curve: ecdsa.curves.Curve =
                                         SECP256k1)

    Bases: object

    curve

    static decode (addr: Optional[str]) → bytes
    static encode (addr: Optional[bytes]) → str
    public_key

    value

class aergo.herapy.obj.address.GovernanceTxAddress
    Bases: enum.Enum
    An enumeration.

    ENTERPRISE = 'aergo.enterprise'
    NAME = 'aergo.name'
    SYSTEM = 'aergo.system'

aergo.herapy.obj.address.check_name_address (addr: str) → int
```

aergo.herapy.obj.aer module

```
class aergo.herapy.obj.aer.Aer (value: Union[bytes, str, int, float] = '0 aer')
    Bases: object
    Return Aergo Unit, AER(//).

    aer

    aergo

    dec

    gaer
```

aergo.herapy.obj.aergo_conf module

class aergo.herapy.obj.aergo_conf.**AergoConfig**

Bases: `object`

`account`

`account_unlocktimeout`

`add_conf` (*k*, *v*, *c*='base')

`auth`

`auth_enablelocalconf`

`authdir`

`blockchain`

`blockchain_coinbaseaccount`

`blockchain_forceresetheight`

`blockchain_maxanchorcount`

`blockchain_maxblocksize`

`blockchain_statetrace`

`blockchain_verifiercount`

`blockchain_verifyonly`

`blockchain_zerofee`

`conf`

`consensus`

`consensus_blockinterval`

`consensus_enablebp`

`consensusRAFT`

`datadir`

`dbtype`

`enableprofile`

`enabletestmode`

`mempool`

`mempool_dumpfilepath`

`mempool_enablefadeout`

`mempool_fadeoutperiod`

`mempool_showmetrics`

`mempool_verifiers`

`monitor`

`monitor_endpoint`

`monitor_protocol`

p2p
p2p_logfullpeerid
p2p_netprotocoladdr
p2p_netprotocolport
p2p_npaddpeers
p2p_npaddpolarises
p2p_npbindaddr
p2p_npbindport
p2p_npcert
p2p_npdiscoverpeers
p2p_npexposeself
p2p_nphiddenpeers
p2p_npkey
p2p_npmaxpeers
p2p_nppeerpool
p2p_nptls
p2p_npusepolaris
personal
polaris
polaris_allowprivate
polaris_genesisfile
profileport
rpc
rpc_netSERVICEaddr
rpc_netSERVICEport
rpc_netSERVICetrace
rpc_nsallowcors
rpc_nscacert
rpc_nscert
rpc_nskey
rpc_nstls
usetestnet

aergo.herapy.obj.block module

```
class aergo.herapy.obj.block.Block (hash_value: Union[aergo.herapy.obj.block_hash.BlockHash,  
str, bytes, None] = None, height: Optional[int] = None,  
grpc_block=None, grpc_block_header=None, tx_cnt: int  
= 0, size: int = 0)
```

Bases: `object`

`block_no`

`blocks_root_hash`

`chain_id`

`chain_id_hash`

`chain_id_hash_b58`

`coinbase_account`

`confirms`

`datetimestamp`

`get_tx` (index: int) → aergo.herapy.obj.transaction.Transaction

`hash`

`height`

`json` (header_only: bool = False) → Dict[KT, VT]

`num_of_tx`

`prev`

`public_key`

`receipts_root_hash`

`sign`

`size`

`timestamp`

`tx_list`

`txs_root_hash`

aergo.herapy.obj.block_hash module

```
class aergo.herapy.obj.block_hash.BlockHash (bh: Union[str, bytes])
```

Bases: `object`

`value`

aergo.herapy.obj.block_meta_stream module

```
class aergo.herapy.obj.block_meta_stream.BlockMetaStream (block_meta_stream)
```

Bases: `aergo.herapy.obj.stream.Stream`

aergo.herapy.obj.block_stream module

```
class aergo.herapy.obj.block_stream.BlockStream(block_stream)
    Bases: aergo.herapy.obj.stream.Stream
    next() → aergo.herapy.obj.block.Block
```

aergo.herapy.obj.blockchain_info module

```
class aergo.herapy.obj.blockchain_info.BlockchainInfo(chain_info,          consen-
                                                    sus_info=None)
    Bases: object
    consensus_info
    gas_price
    json()
    max_block_size
    max_tokens
    minimum_staking
    name_price
    number_of_bp
    total_staking
```

aergo.herapy.obj.blockchain_status module

```
class aergo.herapy.obj.blockchain_status.BlockchainStatus(status)
    Bases: object
    best_block_hash
    best_block_height
    best_chain_id_hash
    best_chain_id_hash_b58
    consensus_info
    json() → Dict[KT, VT]
```

aergo.herapy.obj.call_info module

```
class aergo.herapy.obj.call_info.CallInfo(name, args)
    Bases: object
    CallInfo is used to store contract call/query arguments for json serialization.
```

aergo.herapy.obj.chain_id module

```
class aergo.herapy.obj.chain_id.ChainID(chain_id)
    Bases: object

    consensus
    is_mainnet
    is_public
    json()
    magic
```

aergo.herapy.obj.change_conf_info module

```
class aergo.herapy.obj.change_conf_info.ChangeConfInfo(info)
    Bases: object

    ChangeConfInfo shows the state of the request 'changeCluster' to change configuration of RAFT cluster and
    member list of the cluster.

    error
    json()
    members
    state

class aergo.herapy.obj.change_conf_info.ChangeConfState
    Bases: enum.Enum

    ChangeConfState holds the state of the request 'changeCluster' to change configuration of RAFT cluster.

    APPLIED = 2
    PROPOSED = 0
    SAVED = 1
```

aergo.herapy.obj.consensus_info module

```
class aergo.herapy.obj.consensus_info.ConsensusInfo(info, consensus_type=None)
    Bases: object

    block_producer_list
    detail
    json()
    lib_hash
        get the last irreversible block (LIB) hash :return:
    lib_no
        get the last irreversible block (LIB) number :return:
    status
    type
```

aergo.herapy.obj.event module

```
class aergo.herapy.obj.event.Event (grpc_event)
    Bases: object
    arguments
    block_hash
    block_height
    contract_address
    index
    json ()
    name
    tx_hash
    tx_index
```

aergo.herapy.obj.event_stream module

```
class aergo.herapy.obj.event_stream.EventStream (event_stream)
    Bases: aergo.herapy.obj.stream.Stream
    next ()
```

aergo.herapy.obj.name_info module

```
class aergo.herapy.obj.name_info.NameInfo (info)
    Bases: object
    NameInfo is used to store information of name system.
    destination
    info
    json ()
    name
    owner
```

aergo.herapy.obj.node_info module

```
class aergo.herapy.obj.node_info.NodeInfo (node_info)
    Bases: object
    json ()
```


aergo.herapy.obj.peer module

```

class aergo.herapy.obj.peer.Peer
    Bases: object
    address
    id
    info
    json()
    port
    state

```

aergo.herapy.obj.private_key module

```

class aergo.herapy.obj.private_key.PrivateKey(pk: Union[str, bytes, None])
    Bases: object
    address
    asymmetric_decrypt_msg(address: Union[str, aergo.herapy.obj.address.Address], enc_msg:
        Union[str, bytes]) → bytes
    asymmetric_encrypt_msg(address: Union[str, aergo.herapy.obj.address.Address], msg:
        Union[str, bytes]) → str
    get_signing_key() → ecdsa.keys.SigningKey
    public_key
    sign_msg(msg: bytes) → bytes
    verify_sign(msg: bytes, sign: bytes) → bool

```

aergo.herapy.obj.sc_state module

```

class aergo.herapy.obj.sc_state.SCState(account: aergo.herapy.account.Account,
    var_proofs: aergo.herapy.obj.var_proof.VarProofs)
    Bases: object
    SCState holds the inclusion/exclusion proofs of a contract state in the global trie and of a variable's value in the
    contract trie. SCState is returned by aergo.query_sc_state() for easy merkle proof verification give a root.
    account
    var_proofs
    verify_proof(root: bytes) → bool
        Verify that the given inclusion and exclusion proofs are correct

```

```

class aergo.herapy.obj.sc_state.SCStateVar(var_name: str, array_index: Optional[int] =
    None, map_key: Optional[str] = None, empty:
    bool = False)
    Bases: object
    SCStateVar represents each variable of a calling smart contract. If the variable is the 'state.var' type, you can
    skip 'array_index' and 'map_key'. If the variable is the 'state.array' type, use 'array_index' with the index
    number. If the variable is the 'state.map' type, use 'map_key' with the key name of the map.

```

aergo.herapy.obj.stream module

```
class aergo.herapy.obj.stream.Stream(grpc_stream)
    Bases: object

    cancel()
    cancelled()
    done()
    is_active()
    next()
    running()
    start()
    started
    stop()
    stopped
```

aergo.herapy.obj.transaction module

Transaction class.

```
class aergo.herapy.obj.transaction.Transaction(from_address: Union[bytes,
aergo.herapy.obj.address.Address,
None] = None,
to_address: Union[bytes,
aergo.herapy.obj.address.Address,
None] = None, nonce: int = 0, amount:
Union[bytes, str, int, float] = 0, payload:
Optional[bytes] = None, gas_price: int
= 0, gas_limit: int = 0, read_only: bool
= False, tx_hash: Optional[bytes] =
None, tx_sign: Union[bytes, str, None]
= None, tx_type=<TxType.TRANSFER:
4>, chain_id: Optional[bytes] = None,
block=None, index_in_block: int = -1,
is_in_mempool: bool = False)
```

Bases: `object`

Transaction data structure.

```
amount
block
calculate_hash(including_sign: bool = True) → bytes
chain_id
from_address
gas_limit
gas_price
index_in_block
```

```
is_in_mempool
json (without_block: bool = False) → Dict[KT, VT]
nonce
payload
payload_str
sign
sign_str
to_address
tx_hash
tx_type
```

```
class aergo.herapy.obj.transaction.TxType
    Bases: enum.Enum
    An enumeration.
    GOVERNANCE = 1
    NORMAL = 0
    SC_CALL = 5
    SC_DEPLOY = 6
    SC_FEE_DELEGATION = 3
    SC_REDEPLOY = 2
    TRANSFER = 4
```

aergo.herapy.obj.tx_hash module

```
class aergo.herapy.obj.tx_hash.TxHash (th: Optional[bytes])
    Bases: object
```

aergo.herapy.obj.tx_result module

```
class aergo.herapy.obj.tx_result.TxResult (result, tx=None)
    Bases: object
    json ()
    type
class aergo.herapy.obj.tx_result.TxResultType
    Bases: enum.Enum
    An enumeration.
    COMMIT_RESULT = 0
    RECEIPT = 1
```

aergo.herapy.obj.var_proof module

```
class aergo.herapy.obj.var_proof.VarProofs (var_proofs, storage_keys: List[bytes])
    Bases: list

    VarProof holds the inclusion/exclusion proof of a variable state inside a contract state trie

    storage_keys
    var_proofs

    verify_proof (root: bytes) → bool
        verify that the given inclusion and exclusion proofs are correct

    verify_var_proof (root: bytes, var_proof, trie_key: bytes) → bool
```

Module contents

aergo.herapy.status package

Submodules

aergo.herapy.status.commit_status module

Enumeration of Commit Status.

```
class aergo.herapy.status.commit_status.CommitStatus
    Bases: enum.IntEnum

    TX_OK = 0 TX_NONCE_TOO_LOW = 1 TX_ALREADY_EXISTS = 2 TX_INVALID_HASH
    = 3 TX_INVALID_SIGN = 4 TX_INVALID_FORMAT = 5 TX_INSUFFICIENT_BALANCE = 6
    TX_HAS_SAME_NONCE = 7 TX_INTERNAL_ERROR = 9

    TX_ALREADY_EXISTS = 2
    TX_HAS_SAME_NONCE = 7
    TX_INSUFFICIENT_BALANCE = 6
    TX_INTERNAL_ERROR = 9
    TX_INVALID_FORMAT = 5
    TX_INVALID_HASH = 3
    TX_INVALID_SIGN = 4
    TX_NONCE_TOO_LOW = 1
    TX_OK = 0
```

aergo.herapy.status.peer_status module

Enumeration of Smart contract Status.

```
class aergo.herapy.status.peer_status.PeerStatus
    Bases: enum.Enum

    github.com/aergoio/aergo/types/peerstate.go
```

```

DOWN = 3
HANDSHAKING = 1
RUNNING = 2
STARTING = 0
STOPPED = 4

```

aergo.herapy.status.tx_result_status module

Enumeration of Smart contract Status.

```

class aergo.herapy.status.tx_result_status.TxResultStatus
    Bases: enum.Enum

    An enumeration.

    CREATED = 'CREATED'
    ERROR = 'ERROR'
    SUCCESS = 'SUCCESS'

```

Module contents

aergo.herapy.utils package

Submodules

aergo.herapy.utils.converter module

Common utility module for converting types.

```

aergo.herapy.utils.converter.bigint_to_bytes(v: int) → bytes
aergo.herapy.utils.converter.bytes_to_int_str(v)
aergo.herapy.utils.converter.bytes_to_public_key(v, curve=SECP256k1)
aergo.herapy.utils.converter.convert_aergo_conf_to_toml(aergo_conf:
                                                         aergo.herapy.obj.aergo_conf.AergoConfig)
                                                         → str
aergo.herapy.utils.converter.convert_bigint_to_bytes(number: int) → bytes
aergo.herapy.utils.converter.convert_bytes_to_hex_str(v)
aergo.herapy.utils.converter.convert_bytes_to_int_str(v)
aergo.herapy.utils.converter.convert_bytes_to_public_key(v: bytes, curve:
                                                         ecdsa.curves.Curve
                                                         = SECP256k1) →
                                                         ecdsa.ecdsa.Public_key
aergo.herapy.utils.converter.convert_ip_bytes_to_str(ip)
aergo.herapy.utils.converter.convert_public_key_to_bytes(pubkey,
                                                         curve=SECP256k1,
                                                         compressed=True) →
                                                         bytes

```

```
aergo.herapy.utils.converter.convert_toml_to_aergo_conf (v: str) →  
aergo.herapy.obj.aergo_conf.AergoConfig  
aergo.herapy.utils.converter.convert_tx_to_formatted_json (tx)  
aergo.herapy.utils.converter.convert_tx_to_grpc_tx (tx)  
aergo.herapy.utils.converter.convert_tx_to_json (tx)  
aergo.herapy.utils.converter.get_hash (*strings, no_rand: bool = False, no_encode: bool =  
False) → Union[str, bytes, None]  
aergo.herapy.utils.converter.privkey_to_address (privkey: bytes, curve:  
ecdsa.curves.Curve = SECP256k1,  
compressed: bool = True) → str  
aergo.herapy.utils.converter.public_key_to_bytes (pubkey, curve=SECP256k1, com-  
pressed=True)  
aergo.herapy.utils.converter.tx_to_formatted_json (v)  
aergo.herapy.utils.converter.tx_to_grpc_tx (v)  
aergo.herapy.utils.converter.tx_to_json (v)
```

aergo.herapy.utils.encoding module

```
aergo.herapy.utils.encoding.decode_address (address: str) → bytes  
aergo.herapy.utils.encoding.decode_b58 (v: Union[str, bytes, None]) → Optional[bytes]  
aergo.herapy.utils.encoding.decode_b58_check (v: Union[str, bytes, None]) → Op-  
tional[bytes]  
aergo.herapy.utils.encoding.decode_b64 (v)  
aergo.herapy.utils.encoding.decode_block_hash (block_hash: str) → Optional[bytes]  
aergo.herapy.utils.encoding.decode_payload (payload_str)  
aergo.herapy.utils.encoding.decode_private_key (private_key: Optional[str]) → Op-  
tional[bytes]  
aergo.herapy.utils.encoding.decode_public_key (public_key, curve=SECP256k1)  
aergo.herapy.utils.encoding.decode_root (root: Union[str, bytes, None]) → Optional[bytes]  
aergo.herapy.utils.encoding.decode_signature (sign: Optional[str]) → Optional[bytes]  
aergo.herapy.utils.encoding.decode_tx_hash (tx_hash: Union[str, bytes, None]) → Op-  
tional[bytes]  
aergo.herapy.utils.encoding.encode_address (address: bytes) → str  
aergo.herapy.utils.encoding.encode_b58 (v: Union[str, bytes, None]) → Optional[str]  
aergo.herapy.utils.encoding.encode_b58_check (v: Union[str, bytes, None]) → Op-  
tional[str]  
aergo.herapy.utils.encoding.encode_b64 (v)  
aergo.herapy.utils.encoding.encode_block_hash (block_hash: bytes) → Optional[str]  
aergo.herapy.utils.encoding.encode_payload (payload: Union[str, bytes, None]) → Op-  
tional[str]  
aergo.herapy.utils.encoding.encode_private_key (private_key: bytes) → Optional[str]
```

```
aergo.herapy.utils.encoding.encode_signature (sign: Optional[bytes]) → Optional[str]
aergo.herapy.utils.encoding.encode_tx_hash (tx_hash: Optional[bytes]) → Optional[str]
aergo.herapy.utils.encoding.is_empty (v: Union[str, bytes, None]) → bool
```

aergo.herapy.utils.merkle_proof module

```
aergo.herapy.utils.merkle_proof.bit_is_set (bits: bytes, i: int) → bool
aergo.herapy.utils.merkle_proof.verify_exclusion (root: bytes, ap: List[bytes], key:
                                                bytes, proofKey: bytes, proofVal:
                                                bytes) → bool
    verify_exclusion verifies the merkle proof that a default node (bytes([0]) is included on the path of the 'key', or
    that the proofKey/proofVal key pair is included on the path of the 'key'
aergo.herapy.utils.merkle_proof.verify_exclusion_c (root: bytes, ap: List[bytes],
                                                    length: int, bitmap: bytes, key:
                                                    bytes, proofKey: bytes, proofVal:
                                                    bytes) → bool
    verify_exclusion_c verifies the compressed merkle proof that a default node (bytes([0]) is included on the path
    of the 'key', or that the proofKey/proofVal key pair is included on the path of the 'key'
aergo.herapy.utils.merkle_proof.verify_inclusion (ap: List[bytes], root: bytes, key:
                                                bytes, value: bytes) → bool
    verify_inclusion verifies the merkle proof 'ap' (audit path) that the key/value pair in included in the trie with
    root 'root'.
aergo.herapy.utils.merkle_proof.verify_inclusion_c (ap: List[bytes], height: int,
                                                    bitmap: bytes, root: bytes, key:
                                                    bytes, value: bytes) → bool
    verify_inclusion verifies the compressed merkle proof 'ap' (audit path) that the key/value pair in included in the
    trie with root 'root'.
aergo.herapy.utils.merkle_proof.verify_proof (ap: List[bytes], key_index: int, key: bytes,
                                                leaf_hash: bytes) → bytes
    verify_proof recursively hashes the result with the proof nodes in the audit path 'ap'
aergo.herapy.utils.merkle_proof.verify_proof_c (bitmap: bytes, key: bytes, leaf_hash:
                                                bytes, ap: List[bytes], length: int,
                                                key_index: int, ap_index: int) → bytes
    verify_proof_c recursively hashes the result with the proof nodes in the compressed audit path 'ap'
```

aergo.herapy.utils.signature module

```
aergo.herapy.utils.signature.canonicalize_int (n, order)
aergo.herapy.utils.signature.deserialize_sig (sig)
aergo.herapy.utils.signature.serialize_sig (r, s, order) → bytes
aergo.herapy.utils.signature.uncompress_key (compressed_key_hex)
    base source : https://stackoverflow.com/questions/43629265/deriving-an-ecdsa-uncompressed-public-key-from-a-compressed-one?rq=1 The code from bitcointalk sometimes produces a hex string uncompressed key
    of uneven length.
aergo.herapy.utils.signature.verify_sig (msg, sig, address)
    Verify that the signature 'sig' of the message 'msg' was made by 'address'
```

Module contents

3.1.2 Submodules

3.1.3 aergo.herapy.account module

```
class aergo.herapy.account.Account (private_key: Union[str, bytes, None] = None, empty: bool  
                                     = False)  
    Bases: object  
    Account can be a user account with private and public key, or a contract account.  
    address  
    balance  
    code_hash  
    static decrypt_account (encrypted_bytes: bytes, password: Union[str, bytes]) →  
                             aergo.herapy.account.Account  
    https://cryptography.io/en/latest/hazmat/primitives/aead/ :param encrypted_bytes: encrypted data (bytes)  
    of account :param password: to decrypt the exported bytes :return: account instance  
    static decrypt_from_keystore (keystore: Union[Dict[KT, VT], str], password: str) →  
                                   aergo.herapy.account.Account  
    static encrypt_account (account: aergo.herapy.account.Account, password: Union[str, bytes])  
                             → bytes  
    https://cryptography.io/en/latest/hazmat/primitives/aead/ :param account: account to export :return: en-  
    crypt account data (bytes)  
    static encrypt_to_keystore (account: aergo.herapy.account.Account, password: str, kdf_n:  
                               int = 262144) → Dict[KT, VT]  
    static from_json (data: Union[Dict[KT, VT], str], password: Union[bytes, str, None] = None) →  
                      aergo.herapy.account.Account  
    json (password: Union[bytes, str, None] = None, with_private_key: bool = False) → Dict[KT, VT]  
    nonce  
    private_key  
    public_key  
    sign_msg_hash (msg_hash: bytes) → Optional[bytes]  
    sql_recovery_point  
    state  
    state_proof  
    storage_root  
    verify_proof (root: Union[str, bytes]) → bool  
    verify that the given inclusion and exclusion proofs are correct  
    verify_sign (msg_hash: bytes, sign: bytes) → Optional[bool]
```

3.1.4 aergo.herapy.aergo module

Main module.


```

class aergo.herapy.aergo.Aergo
    Bases: object

    Main class for herapy

    account
        Returns the account object. :return:

    batch_call_sc (sc_txs: List[aergo.herapy.obj.transaction.Transaction])
        → Tuple[List[aergo.herapy.obj.transaction.Transaction],
        List[aergo.herapy.obj.tx_result.TxResult]]

    batch_tx (signed_txs: List[aergo.herapy.obj.transaction.Transaction]) → Tuple[
        List[aergo.herapy.obj.transaction.Transaction], List[aergo.herapy.obj.tx_result.TxResult]]
        Send a set of signed transactions simultaneously. These transactions will push to the memory pool after
        verifying. :param signed_txs: :return:

    call_sc (sc_address: Union[str, aergo.herapy.obj.address.GovernanceTxAddress, bytes], func_name:
        str, amount: int = 0, args: Optional[Any] = None, gas_limit: int = 0, gas_price: int = 0) →
        Tuple[aergo.herapy.obj.transaction.Transaction, aergo.herapy.obj.tx_result.TxResult]

    connect (target: str, tls_ca_cert: Optional[str] = None, tls_cert: Optional[str] = None, tls_key: Op-
        tional[str] = None) → None
        Connect to the gRPC server running on port target e.g. target="localhost:7845". :param target: :param
        tls_ca_cert: :param tls_cert: :param tls_key: :return:

    deploy_sc (payload: Union[str, bytes], amount: Union[bytes, str, int, float] = 0, args: Optional[Any]
        = None, retry_nonce: int = 0, redeploy: bool = False, gas_limit: int = 0, gas_price: int =
        0)

    disconnect () → None
        Disconnect from the gRPC server.

    export_account (password: Union[str, bytes], account: Optional[aergo.herapy.account.Account] =
        None) → str

    export_account_to_keystore (password: str, account: Optional[aergo.herapy.account.Account] = None, kdf_n: int =
        262144) → Dict[KT, VT]

    export_account_to_keystore_file (keystore_path: str, password: str, account: Op-
        tional[aergo.herapy.account.Account] = None, kdf_n: int
        = 262144) → None

    generate_tx (to_address: Union[bytes, str, None], nonce: int, amount: Union[bytes, str, int,
        float], gas_limit: int = 0, gas_price: int = 0, payload: Optional[bytes] =
        None, tx_type: aergo.herapy.obj.transaction.TxType = <TxType.NORMAL: 0>) →
        aergo.herapy.obj.transaction.Transaction

    get_abi (contract_addr: str = None, addr_bytes: bytes = None)
        Returns the abi of given contract address.

    get_account (account: Optional[aergo.herapy.account.Account] = None, address: Union[str, bytes,
        aergo.herapy.obj.address.Address, None] = None, proof: bool = False, root: bytes =
        b'', compressed: bool = True) → aergo.herapy.account.Account
        Return account information :param address: :param proof: :param root: :param compressed: :return:

    get_address (account: Optional[aergo.herapy.account.Account] = None) → Op-
        tional[aergo.herapy.obj.address.Address]

    get_block (block_hash: Union[bytes, aergo.herapy.obj.block_hash.BlockHash, None] = None,
        block_height: int = -1) → aergo.herapy.obj.block.Block
        Returns block information for block_hash or block_height. :param block_hash: :param block_height:
        :return:

```

get_block_headers (*block_hash: Optional[bytes] = None, block_height: int = -1, list_size: int = 20, offset: int = 0, is_asc_order: bool = False*) → List[aergo.herapy.obj.block.Block]
Returns the list of blocks. :param block_hash: :param block_height: :param list_size: maximum number of results :param offset: the start point to search until the block_hash or block_height :param is_asc_order: :return:

get_block_meta (*block_hash: Union[bytes, aergo.herapy.obj.block_hash.BlockHash, None] = None, block_height: int = -1*) → aergo.herapy.obj.block.Block
Returns block metadata for *block_hash* or *block_height*. :param block_hash: :param block_height: :return:

get_block metas (*block_hash: Optional[bytes] = None, block_height: int = -1, list_size: int = 20, offset: int = 0, is_asc_order: bool = False*) → List[aergo.herapy.obj.block.Block]
Returns the list of metadata of queried blocks. :param block_hash: :param block_height: :param list_size: maximum number of results :param offset: the start point to search until the block_hash or block_height :param is_asc_order: :return:

get_blockchain_status () → Tuple[aergo.herapy.obj.block_hash.BlockHash, int]
Returns the highest block hash and block height so far. :return:

get_chain_info (*with_consensus_info: bool = True*) → aergo.herapy.obj.blockchain_info.BlockchainInfo
Returns the blockchain info :return:

get_conf_change_progress (*block_height: int*) → aergo.herapy.obj.change_conf_info.ChangeConfInfo
Returns the RAFT change config progress status after 'changeCluster' system contract :return:

get_consensus_info () → aergo.herapy.obj.consensus_info.ConsensusInfo
Returns the consensus information :return:

get_enterprise_config (*key: str*) → aergo.herapy.obj.enterprise_config.EnterpriseConfig

get_events (*sc_address: Union[bytes, str, aergo.herapy.obj.tx_hash.TxHash], event_name: str, start_block_no: int = -1, end_block_no: int = -1, with_desc: bool = False, arg_filter: Union[str, Dict[KT, VT], List[T], Tuple, None] = None, recent_block_cnt: int = 0*) → List[aergo.herapy.obj.event.Event]

get_name_info (*name: str, block_height: int = -1*)
Returns information of name which is designated by the system contract :param name: :param block_height: :return:

get_node_accounts (*skip_state: bool = False*) → List[aergo.herapy.account.Account]
Returns a list of all node accounts. :return:

get_node_info (*keys: Optional[str] = None*) → aergo.herapy.obj.node_info.NodeInfo
Returns the consensus information :return:

get_node_state (*timeout: int = 1*) → Dict[KT, VT]
Returns information about the node state. :return:

get_peers () → List[aergo.herapy.obj.peer.Peer]
Returns a list of peers. :return:

get_status () → aergo.herapy.obj.blockchain_status.BlockchainStatus
Returns the blockchain status :return:

get_tx (*tx_hash: Union[str, aergo.herapy.obj.tx_hash.TxHash, bytes], mempool_only: bool = False, skip_block: bool = False*) → aergo.herapy.obj.transaction.Transaction
Returns info on transaction with hash *tx_hash*. :param tx_hash: :return:

get_tx_result (*tx_hash: Union[str, aergo.herapy.obj.tx_hash.TxHash, bytes]*) → aergo.herapy.obj.tx_result.TxResult

import_account (*exported_data: Union[str, bytes], password: Union[str, bytes], skip_state: bool = False, skip_self: bool = False*) → aergo.herapy.account.Account

```

import_account_from_keystore (keystore: Union[Dict[KT, VT], str], password: str,
                               skip_state: bool = False, skip_self: bool = False) →
                               aergo.herapy.account.Account

import_account_from_keystore_file (keystore_path: str, password: str, skip_state:
                                     bool = False, skip_self: bool = False) →
                                     aergo.herapy.account.Account

lock_account (address: bytes, passphrase: str)
    Locks the account with address address with the passphrase passphrase. :param address: :param
    passphrase: :return:

new_account (private_key: Union[str, bytes, None] = None, skip_state: bool = False) →
    aergo.herapy.account.Account

new_call_sc_tx (sc_address: Union[str, aergo.herapy.obj.address.GovernanceTxAddress, bytes],
                 func_name: str, amount: int = 0, args: Optional[Any] = None, nonce:
                 Optional[int] = None, gas_limit: int = 0, gas_price: int = 0) →
                 aergo.herapy.obj.transaction.Transaction

query_sc (sc_address: Union[bytes, str], func_name: str, args: Optional[Any] = None)

query_sc_state (sc_address: Union[bytes, str], storage_keys: List[Union[bytes, str,
    aergo.herapy.obj.sc_state.SCStateVar]], root: bytes = b'', compressed: bool
    = True) → aergo.herapy.obj.sc_state.SCState
    query_sc_state returns a SCState object containing the contract state and variable state with their respective
    merkle proofs.

receive_block_meta_stream () → aergo.herapy.obj.block_stream.BlockStream
    Returns the iterable block stream :return:

receive_block_stream () → aergo.herapy.obj.block_stream.BlockStream
    Returns the iterable block stream :return:

receive_event_stream (sc_address: Union[str, bytes, aergo.herapy.obj.tx_hash.TxHash],
                       event_name: str, start_block_no: int = 0, end_block_no: int =
                       0, with_desc: bool = False, arg_filter: Union[str, Dict[KT, VT],
                       List[T], Tuple, None] = None, recent_block_cnt: int = 0) →
                       aergo.herapy.obj.event_stream.EventStream

send_payload (amount: Union[bytes, str, int, float], payload: Optional[bytes] =
    None, to_address: Union[str, bytes, aergo.herapy.obj.address.Address,
    aergo.herapy.obj.address.GovernanceTxAddress, None] = None, retry_nonce: int = 0,
    tx_type: aergo.herapy.obj.transaction.TxType = <TxType.TRANSFER: 4>, gas_limit:
    int = 0, gas_price: int = 0) → Tuple[aergo.herapy.obj.transaction.Transaction,
    aergo.herapy.obj.tx_result.TxResult]

send_tx (signed_tx: aergo.herapy.obj.transaction.Transaction) → Tu-
    ple[aergo.herapy.obj.transaction.Transaction, aergo.herapy.obj.tx_result.TxResult]
    Send a signed transaction. This transaction will push to the memory pool after verifying. :param signed_tx:
    :return:

send_unsigned_tx (unsigned_tx: aergo.herapy.obj.transaction.Transaction)
    Sends the unsigned transaction. The unsigned transaction will be signed by the account which is stored in
    the connected node. :param unsigned_tx: :return:

transfer (to_address: Union[str, bytes, aergo.herapy.obj.address.Address,
    aergo.herapy.obj.address.GovernanceTxAddress], amount: Union[bytes, str, int,
    float], retry_nonce: int = 3) → Tuple[aergo.herapy.obj.transaction.Transaction,
    aergo.herapy.obj.tx_result.TxResult]

```

```
unlock_account (address: bytes, passphrase: str)
    Unlocks the account with address address with the passphrase passphrase. :param address: :param
    passphrase: :return:

wait_tx_result (tx_hash: Union[str, aergo.herapy.obj.tx_hash.TxHash, bytes], timeout: int = 30,
    tempo: float = 0.2) → aergo.herapy.obj.tx_result.TxResult
```

3.1.5 aergo.herapy.comm module

Communication(grpc) module.

```
class aergo.herapy.comm.Comm(target: Optional[str] = None, tls_ca_cert: Optional[bytes] =
    None, tls_cert: Optional[bytes] = None, tls_key: Optional[bytes]
    = None)

    Bases: object

add_raft_member (request_id: int, member_id: int, member_name: str, member_address: str, mem-
    ber_peer_id: bytes)

commit_txs (signed_txs: List[aergo.herapy.obj.transaction.Transaction])

connect ()

create_account (address: bytes, passphrase: str)

del_raft_member (request_id: int, member_id: int, member_name: str, member_address: str, mem-
    ber_peer_id: bytes)

disconnect ()

get_abi (addr_bytes: bytes)

get_account_state (address: bytes)

get_account_state_proof (address: bytes, root: bytes, compressed: bool)

get_accounts ()

get_block (query: bytes)

get_block_headers (block_hash: Optional[bytes], block_height: int, list_size: int, offset: int,
    is_asc_order: bool)

get_block_meta (query: bytes)

get_block metas (block_hash: Optional[bytes], block_height: int, list_size: int, offset: int,
    is_asc_order: bool)

get_block_tx (tx_hash: bytes)

get_blockchain_status ()

get_chain_info ()

get_conf_change_progress (block_height: int)

get_consensus_info ()

get_enterprise_config (key: str)

get_events (sc_address: bytes, event_name: str, start_block_no: int, end_block_no: int, with_desc:
    bool, arg_filter: Optional[bytes], recent_block_cnt: int)

get_name_info (name: str, block_height: int)

get_node_info (keys: Optional[str])
```

```
get_node_state (timeout: int)
get_peers ()
get_receipt (tx_hash: bytes)
get_tx (tx_hash: bytes)
lock_account (address: bytes, passphrase: str)
query_contract (sc_address: bytes, query_info: bytes)
query_contract_state (sc_address: bytes, storage_keys: List[bytes], root: bytes, compressed:
                        bool)
receive_block_meta_stream()
receive_block_stream()
receive_event_stream (sc_address: bytes, event_name: str, start_block_no: int, end_block_no:
                        int, with_desc: bool, arg_filter: Optional[bytes], recent_block_cnt: int)
send_tx (unsigned_tx: aergo.herapy.obj.transaction.Transaction)
unlock_account (address: bytes, passphrase: str)
```

3.1.6 aergo.herapy.constants module

3.1.7 Module contents

Top-level package for herapy.

3.2 Examples and projects using Herapy

- <https://github.com/aergoio/eth-merkle-bridge>
- <https://github.com/aergoio/merkle-bridge>
- Aergo private enterprise
- https://github.com/aergoio/herapy/tree/develop/tests/test_integration

CHAPTER 4

Indices and tables

- `genindex`
- `modindex`
- `search`

a

aergo.herapy, 33
aergo.herapy.account, 28
aergo.herapy.aergo, 28
aergo.herapy.comm, 32
aergo.herapy.constants, 33
aergo.herapy.errors, 10
aergo.herapy.errors.conversion_exception, 9
aergo.herapy.errors.exception, 9
aergo.herapy.errors.general_exception, 10
aergo.herapy.errors.InsufficientBalanceError, 9
aergo.herapy.grpc, 13
aergo.herapy.grpc.account_pb2, 10
aergo.herapy.grpc.account_pb2_grpc, 10
aergo.herapy.grpc.blockchain_pb2, 10
aergo.herapy.grpc.blockchain_pb2_grpc, 10
aergo.herapy.grpc.metric_pb2, 10
aergo.herapy.grpc.metric_pb2_grpc, 10
aergo.herapy.grpc.node_pb2, 10
aergo.herapy.grpc.node_pb2_grpc, 10
aergo.herapy.grpc.p2p_pb2, 10
aergo.herapy.grpc.p2p_pb2_grpc, 10
aergo.herapy.grpc.pmap_pb2, 10
aergo.herapy.grpc.pmap_pb2_grpc, 10
aergo.herapy.grpc.polarrpc_pb2, 10
aergo.herapy.grpc.polarrpc_pb2_grpc, 10
aergo.herapy.grpc.raft_pb2, 11
aergo.herapy.grpc.raft_pb2_grpc, 11
aergo.herapy.grpc.rpc_pb2, 11
aergo.herapy.grpc.rpc_pb2_grpc, 11
aergo.herapy.obj, 24
aergo.herapy.obj.abi, 14
aergo.herapy.obj.address, 14
aergo.herapy.obj.aer, 14
aergo.herapy.obj.aergo_conf, 15
aergo.herapy.obj.block, 17
aergo.herapy.obj.block_hash, 17
aergo.herapy.obj.block_meta_stream, 17
aergo.herapy.obj.block_stream, 18
aergo.herapy.obj.blockchain_info, 18
aergo.herapy.obj.blockchain_status, 18
aergo.herapy.obj.call_info, 18
aergo.herapy.obj.chain_id, 19
aergo.herapy.obj.change_conf_info, 19
aergo.herapy.obj.consensus_info, 19
aergo.herapy.obj.event, 20
aergo.herapy.obj.event_stream, 20
aergo.herapy.obj.name_info, 20
aergo.herapy.obj.node_info, 20
aergo.herapy.obj.peer, 21
aergo.herapy.obj.private_key, 21
aergo.herapy.obj.sc_state, 21
aergo.herapy.obj.stream, 22
aergo.herapy.obj.transaction, 22
aergo.herapy.obj.tx_hash, 23
aergo.herapy.obj.tx_result, 23
aergo.herapy.obj.var_proof, 24
aergo.herapy.status, 25
aergo.herapy.status.commit_status, 24
aergo.herapy.status.peer_status, 24
aergo.herapy.status.tx_result_status, 25
aergo.herapy.utils, 28
aergo.herapy.utils.converter, 25
aergo.herapy.utils.encoding, 26
aergo.herapy.utils.merkle_proof, 27
aergo.herapy.utils.signature, 27

A

- Abi (class in *aergo.herapy.obj.abi*), 14
- account (*aergo.herapy.aergo.Aergo* attribute), 29
- account (*aergo.herapy.obj.aergo_conf.AergoConfig* attribute), 15
- account (*aergo.herapy.obj.sc_state.SCState* attribute), 21
- Account (class in *aergo.herapy.account*), 28
- account_unlocktimeout (*aergo.herapy.obj.aergo_conf.AergoConfig* attribute), 15
- add_AergoRPCServiceServicer_to_server() (in module *aergo.herapy.grpc.rpc_pb2_grpc*), 13
- add_conf() (*aergo.herapy.obj.aergo_conf.AergoConfig* method), 15
- add_PolarisRPCServiceServicer_to_server() (in module *aergo.herapy.grpc.polarrpc_pb2_grpc*), 11
- add_raft_member() (*aergo.herapy.comm.Comm* method), 32
- AddBLEntry() (*aergo.herapy.grpc.polarrpc_pb2_grpc.PolarisRPCServiceServicer* method), 10
- address (*aergo.herapy.account.Account* attribute), 28
- address (*aergo.herapy.obj.peer.Peer* attribute), 21
- address (*aergo.herapy.obj.private_key.PrivateKey* attribute), 21
- Address (class in *aergo.herapy.obj.address*), 14
- aer (*aergo.herapy.obj.aer.Aer* attribute), 14
- Aer (class in *aergo.herapy.obj.aer*), 14
- aergo (*aergo.herapy.obj.aer.Aer* attribute), 14
- Aergo (class in *aergo.herapy.aergo*), 28
- aergo.herapy (module), 33
- aergo.herapy.account (module), 28
- aergo.herapy.aergo (module), 28
- aergo.herapy.comm (module), 32
- aergo.herapy.constants (module), 33
- aergo.herapy.errors (module), 10
- aergo.herapy.errors.conversion_exception (module), 9
- aergo.herapy.errors.exception (module), 9
- aergo.herapy.errors.general_exception (module), 10
- aergo.herapy.errors.InsufficientBalanceError (module), 9
- aergo.herapy.grpc (module), 13
- aergo.herapy.grpc.account_pb2 (module), 10
- aergo.herapy.grpc.account_pb2_grpc (module), 10
- aergo.herapy.grpc.blockchain_pb2 (module), 10
- aergo.herapy.grpc.blockchain_pb2_grpc (module), 10
- aergo.herapy.grpc.metric_pb2 (module), 10
- aergo.herapy.grpc.metric_pb2_grpc (module), 10
- aergo.herapy.grpc.node_pb2 (module), 10
- aergo.herapy.grpc.node_pb2_grpc (module), 10
- aergo.herapy.grpc.p2p_pb2 (module), 10
- aergo.herapy.grpc.p2p_pb2_grpc (module), 10
- aergo.herapy.grpc.pmap_pb2 (module), 10
- aergo.herapy.grpc.pmap_pb2_grpc (module), 10
- aergo.herapy.grpc.polarrpc_pb2 (module), 10
- aergo.herapy.grpc.polarrpc_pb2_grpc (module), 10
- aergo.herapy.grpc.raft_pb2 (module), 11
- aergo.herapy.grpc.raft_pb2_grpc (module), 11
- aergo.herapy.grpc.rpc_pb2 (module), 11
- aergo.herapy.grpc.rpc_pb2_grpc (module), 11
- aergo.herapy.obj (module), 24
- aergo.herapy.obj.abi (module), 14
- aergo.herapy.obj.address (module), 14
- aergo.herapy.obj.aer (module), 14

AergoConfig (class in), 15

AergoException, 9

AergoRPCServiceServicer (class in), 11

AergoRPCServiceStub (class in), 13

amount (attribute), 22

APPLIED (attribute), 19

arguments (attribute), 20

asymmetric_decrypt_msg ()

(method), 21

asymmetric_encrypt_msg ()

(method), 21

auth (attribute), 15

auth_enablelocalconf

(attribute), 15

authdir (attribute), 15

B

balance (attribute), 28

batch_call_sc () (method), 29

batch_tx () (method), 29

best_block_hash (attribute), 18

best_block_height

(attribute), 18

best_chain_id_hash

(attribute), 18

best_chain_id_hash_b58

(attribute), 18

bigint_to_bytes () (in module), 25

bit_is_set () (in module), 27

BlackList () (method), 11

block (attribute), 22

Block (class in), 17

block_hash (attribute), 20

block_height (attribute), 20

block_no (attribute), 17

block_producer_list

(attribute), 19

blockchain (attribute), 15

Blockchain () (method), 11

blockchain_coinbaseaccount

(attribute), 15

blockchain_forceresetheight

[\(aergo.herapy.obj.aergo_conf.AergoConfig attribute\), 15](#)
[blockchain_maxanchorcount \(aergo.herapy.obj.aergo_conf.AergoConfig attribute\), 15](#)
[blockchain_maxblocksize \(aergo.herapy.obj.aergo_conf.AergoConfig attribute\), 15](#)
[blockchain_statetrace \(aergo.herapy.obj.aergo_conf.AergoConfig attribute\), 15](#)
[blockchain_verifiercount \(aergo.herapy.obj.aergo_conf.AergoConfig attribute\), 15](#)
[blockchain_verifyonly \(aergo.herapy.obj.aergo_conf.AergoConfig attribute\), 15](#)
[blockchain_zerofee \(aergo.herapy.obj.aergo_conf.AergoConfig attribute\), 15](#)
[BlockchainInfo \(class in aergo.herapy.obj.blockchain_info\), 18](#)
[BlockchainStatus \(class in aergo.herapy.obj.blockchain_status\), 18](#)
[BlockHash \(class in aergo.herapy.obj.block_hash\), 17](#)
[BlockMetaStream \(class in aergo.herapy.obj.block_meta_stream\), 17](#)
[blocks_root_hash \(aergo.herapy.obj.block.Block attribute\), 17](#)
[BlockStream \(class in aergo.herapy.obj.block_stream\), 18](#)
[bytes_to_int_str\(\) \(in module aergo.herapy.utils.converter\), 25](#)
[bytes_to_public_key\(\) \(in module aergo.herapy.utils.converter\), 25](#)

C

[calculate_hash\(\) \(aergo.herapy.obj.transaction.Transaction method\), 22](#)
[call_sc\(\) \(aergo.herapy.aergo.Aergo method\), 29](#)
[CallInfo \(class in aergo.herapy.obj.call_info\), 18](#)
[cancel\(\) \(aergo.herapy.obj.stream.Stream method\), 22](#)
[cancelled\(\) \(aergo.herapy.obj.stream.Stream method\), 22](#)
[canonicalize_int\(\) \(in module aergo.herapy.utils.signature\), 27](#)
[chain_id \(aergo.herapy.obj.block.Block attribute\), 17](#)
[chain_id \(aergo.herapy.obj.transaction.Transaction attribute\), 22](#)
[chain_id_hash \(aergo.herapy.obj.block.Block attribute\), 17](#)
[chain_id_hash_b58 \(aergo.herapy.obj.block.Block attribute\), 17](#)
[ChainID \(class in aergo.herapy.obj.chain_id\), 19](#)
[ChainStat\(\) \(aergo.herapy.grpc.rpc_pb2_grpc.AergoRPCServiceService method\), 11](#)
[ChangeConfInfo \(class in aergo.herapy.obj.change_conf_info\), 19](#)
[ChangeConfState \(class in aergo.herapy.obj.change_conf_info\), 19](#)
[ChangeMembership\(\) \(aergo.herapy.grpc.rpc_pb2_grpc.AergoRPCServiceService method\), 11](#)
[check_name_address\(\) \(in module aergo.herapy.obj.address\), 14](#)
[code_hash \(aergo.herapy.account.Account attribute\), 28](#)
[coinbase_account \(aergo.herapy.obj.block.Block attribute\), 17](#)
[Comm \(aergo.herapy.errors.exception.AergoException attribute\), 9](#)
[Comm \(class in aergo.herapy.comm\), 32](#)
[COMMIT_RESULT \(aergo.herapy.obj.tx_result.TxResultType attribute\), 23](#)
[commit_txs\(\) \(aergo.herapy.comm.Comm method\), 32](#)
[CommitStatus \(class in aergo.herapy.status.commit_status\), 24](#)
[CommitTX\(\) \(aergo.herapy.grpc.rpc_pb2_grpc.AergoRPCServiceService method\), 11](#)
[CommunicationException, 10](#)
[conf \(aergo.herapy.obj.aergo_conf.AergoConfig attribute\), 15](#)
[confirms \(aergo.herapy.obj.block.Block attribute\), 17](#)
[connect\(\) \(aergo.herapy.aergo.Aergo method\), 29](#)
[connect\(\) \(aergo.herapy.comm.Comm method\), 32](#)
[consensus \(aergo.herapy.obj.aergo_conf.AergoConfig attribute\), 15](#)
[consensus \(aergo.herapy.obj.chain_id.ChainID attribute\), 19](#)
[consensus_blockinterval \(aergo.herapy.obj.aergo_conf.AergoConfig attribute\), 15](#)
[consensus_enablebp \(aergo.herapy.obj.aergo_conf.AergoConfig attribute\), 15](#)
[consensus_info \(aergo.herapy.obj.blockchain_info.BlockchainInfo attribute\), 18](#)
[consensus_info \(aergo.herapy.obj.blockchain_status.BlockchainStatus attribute\), 18](#)
[consensus_raft \(aergo.herapy.obj.aergo_conf.AergoConfig attribute\), 15](#)
[ConsensusInfo \(class in aergo.herapy.obj.consensus_info\), 19](#)
[contract_address \(aergo.herapy.obj.event.Event attribute\), 20](#)
[Conv \(aergo.herapy.errors.exception.AergoException attribute\), 9](#)

tribute), 9

ConversionException, 9

convert_aergo_conf_to_toml() (in module *aergo.herapy.utils.converter*), 25

convert_bigint_to_bytes() (in module *aergo.herapy.utils.converter*), 25

convert_bytes_to_hex_str() (in module *aergo.herapy.utils.converter*), 25

convert_bytes_to_int_str() (in module *aergo.herapy.utils.converter*), 25

convert_bytes_to_public_key() (in module *aergo.herapy.utils.converter*), 25

convert_ip_bytes_to_str() (in module *aergo.herapy.utils.converter*), 25

convert_public_key_to_bytes() (in module *aergo.herapy.utils.converter*), 25

convert_toml_to_aergo_conf() (in module *aergo.herapy.utils.converter*), 26

convert_tx_to_formatted_json() (in module *aergo.herapy.utils.converter*), 26

convert_tx_to_grpc_tx() (in module *aergo.herapy.utils.converter*), 26

convert_tx_to_json() (in module *aergo.herapy.utils.converter*), 26

create_account() (*aergo.herapy.comm.Comm* method), 32

CreateAccount() (*aergo.herapy.grpc.rpc_pb2_grpc.AergoRPCServiceServ* method), 11

CREATED (*aergo.herapy.status.tx_result_status.TxResultStatus* attribute), 25

CurrentList() (*aergo.herapy.grpc.polarrpc_pb2_grpc.PolarisRPCServiceServ* method), 11

curve (*aergo.herapy.obj.address.Address* attribute), 14

decode_payload() (in module *aergo.herapy.utils.encoding*), 26

decode_private_key() (in module *aergo.herapy.utils.encoding*), 26

decode_public_key() (in module *aergo.herapy.utils.encoding*), 26

decode_root() (in module *aergo.herapy.utils.encoding*), 26

decode_signature() (in module *aergo.herapy.utils.encoding*), 26

decode_tx_hash() (in module *aergo.herapy.utils.encoding*), 26

decrypt_account() (*aergo.herapy.account.Account* static method), 28

decrypt_from_keystore() (*aergo.herapy.account.Account* static method), 28

delRAFT_member() (*aergo.herapy.comm.Comm* method), 32

deploy_sc() (*aergo.herapy.aergo.Aergo* method), 29

deserialize_sig() (in module *aergo.herapy.utils.signature*), 27

destination (*aergo.herapy.obj.name_info.NameInfo* attribute), 20

detail (*aergo.herapy.obj.consensus_info.ConsensusInfo* attribute), 19

disconnect() (*aergo.herapy.comm.Comm* method), 32

DOWN (*aergo.herapy.status.peer_status.PeerStatus* attribute), 24

enableprofile (*aergo.herapy.obj.aergo_conf.AergoConfig* attribute), 15

enabletestmode (*aergo.herapy.obj.aergo_conf.AergoConfig* attribute), 15

encode() (*aergo.herapy.obj.address.Address* static method), 14

encode_address() (in module *aergo.herapy.utils.encoding*), 26

encode_b58() (in module *aergo.herapy.utils.encoding*), 26

encode_b58_check() (in module *aergo.herapy.utils.encoding*), 26

encode_b64() (in module *aergo.herapy.utils.encoding*), 26

encode_block_hash() (in module *aergo.herapy.utils.encoding*), 26

encode_payload() (in module *aergo.herapy.utils.encoding*), 26

D

datadir (*aergo.herapy.obj.aergo_conf.AergoConfig* attribute), 15

timestamp (*aergo.herapy.obj.block.Block* attribute), 17

dbtype (*aergo.herapy.obj.aergo_conf.AergoConfig* attribute), 15

dec (*aergo.herapy.obj.aer.Aer* attribute), 14

decode() (*aergo.herapy.obj.address.Address* static method), 14

decode_address() (in module *aergo.herapy.utils.encoding*), 26

decode_b58() (in module *aergo.herapy.utils.encoding*), 26

decode_b58_check() (in module *aergo.herapy.utils.encoding*), 26

decode_b64() (in module *aergo.herapy.utils.encoding*), 26

decode_block_hash() (in module *aergo.herapy.utils.encoding*), 26

E

enableprofile (*aergo.herapy.obj.aergo_conf.AergoConfig* attribute), 15

enabletestmode (*aergo.herapy.obj.aergo_conf.AergoConfig* attribute), 15

encode() (*aergo.herapy.obj.address.Address* static method), 14

encode_address() (in module *aergo.herapy.utils.encoding*), 26

encode_b58() (in module *aergo.herapy.utils.encoding*), 26

encode_b58_check() (in module *aergo.herapy.utils.encoding*), 26

encode_b64() (in module *aergo.herapy.utils.encoding*), 26

encode_block_hash() (in module *aergo.herapy.utils.encoding*), 26

encode_payload() (in module *aergo.herapy.utils.encoding*), 26

`encode_private_key()` (in module `aergo.herapy.utils.encoding`), 26
`encode_signature()` (in module `aergo.herapy.utils.encoding`), 26
`encode_tx_hash()` (in module `aergo.herapy.utils.encoding`), 27
`encrypt_account()` (`aergo.herapy.account.Account` static method), 28
`encrypt_to_keystore()` (`aergo.herapy.account.Account` static method), 28
`ENTERPRISE` (`aergo.herapy.obj.address.GovernanceTxAddress` attribute), 14
`error` (`aergo.herapy.obj.change_conf_info.ChangeConfInfo` attribute), 19
`ERROR` (`aergo.herapy.status.tx_result_status.TxResultStatus` attribute), 25
`Event` (class in `aergo.herapy.obj.event`), 20
`EventStream` (class in `aergo.herapy.obj.event_stream`), 20
`export_account()` (`aergo.herapy.aergo.Aergo` method), 29
`export_account_to_keystore()` (`aergo.herapy.aergo.Aergo` method), 29
`export_account_to_keystore_file()` (`aergo.herapy.aergo.Aergo` method), 29
`ExportAccount()` (`aergo.herapy.grpc.rpc_pb2_grpc.AergoRPCService` method), 11
F
`from_address` (`aergo.herapy.obj.transaction.Transaction` attribute), 22
`from_json()` (`aergo.herapy.account.Account` static method), 28
`functions` (`aergo.herapy.obj.abi.Abi` attribute), 14
G
`gaer` (`aergo.herapy.obj.aer.Aer` attribute), 14
`gas_limit` (`aergo.herapy.obj.transaction.Transaction` attribute), 22
`gas_price` (`aergo.herapy.obj.blockchain_info.BlockchainInfo` attribute), 18
`gas_price` (`aergo.herapy.obj.transaction.Transaction` attribute), 22
`General` (`aergo.herapy.errors.exception.AergoException` attribute), 10
`GeneralException`, 10
`generate_tx()` (`aergo.herapy.aergo.Aergo` method), 29
`get_abi()` (`aergo.herapy.aergo.Aergo` method), 29
`get_abi()` (`aergo.herapy.comm.Comm` method), 32
`get_account()` (`aergo.herapy.aergo.Aergo` method), 29
`get_account_state()` (`aergo.herapy.comm.Comm` method), 32
`get_account_state_proof()` (`aergo.herapy.comm.Comm` method), 32
`get_accounts()` (`aergo.herapy.comm.Comm` method), 32
`get_address()` (`aergo.herapy.aergo.Aergo` method), 29
`get_block()` (`aergo.herapy.aergo.Aergo` method), 29
`get_block()` (`aergo.herapy.comm.Comm` method), 32
`get_block_headers()` (`aergo.herapy.aergo.Aergo` method), 29
`get_block_headers()` (`aergo.herapy.comm.Comm` method), 32
`get_block_meta()` (`aergo.herapy.aergo.Aergo` method), 30
`get_block_meta()` (`aergo.herapy.comm.Comm` method), 32
`get_block_metas()` (`aergo.herapy.aergo.Aergo` method), 30
`get_block_metas()` (`aergo.herapy.comm.Comm` method), 32
`get_block_tx()` (`aergo.herapy.comm.Comm` method), 32
`get_blockchain_status()` (`aergo.herapy.aergo.Aergo` method), 30
`get_blockchain_status()` (`aergo.herapy.comm.Comm` method), 32
`get_chain_info()` (`aergo.herapy.aergo.Aergo` method), 30
`get_chain_info()` (`aergo.herapy.comm.Comm` method), 32
`get_conf_change_progress()` (`aergo.herapy.aergo.Aergo` method), 30
`get_conf_change_progress()` (`aergo.herapy.comm.Comm` method), 32
`get_consensus_info()` (`aergo.herapy.aergo.Aergo` method), 30
`get_consensus_info()` (`aergo.herapy.comm.Comm` method), 32
`get_enterprise_config()` (`aergo.herapy.aergo.Aergo` method), 30
`get_enterprise_config()` (`aergo.herapy.comm.Comm` method), 32
`get_events()` (`aergo.herapy.aergo.Aergo` method), 30
`get_events()` (`aergo.herapy.comm.Comm` method), 32
`get_hash()` (in module `aergo.herapy.utils.converter`), 26
`get_name_info()` (`aergo.herapy.aergo.Aergo` method), 30
`get_name_info()` (`aergo.herapy.comm.Comm` method), 32

```

get_node_accounts() (aergo.herapy.aergo.Aergo method), 30
get_node_info() (aergo.herapy.aergo.Aergo method), 30
get_node_info() (aergo.herapy.comm.Comm method), 32
get_node_state() (aergo.herapy.aergo.Aergo method), 30
get_node_state() (aergo.herapy.comm.Comm method), 32
get_peers() (aergo.herapy.aergo.Aergo method), 30
get_peers() (aergo.herapy.comm.Comm method), 33
get_receipt() (aergo.herapy.comm.Comm method), 33
get_signing_key() (aergo.herapy.obj.private_key.PrivateKey method), 21
get_status() (aergo.herapy.aergo.Aergo method), 30
get_tx() (aergo.herapy.aergo.Aergo method), 30
get_tx() (aergo.herapy.comm.Comm method), 33
get_tx() (aergo.herapy.obj.block.Block method), 17
get_tx_result() (aergo.herapy.aergo.Aergo method), 30
GetABI() (aergo.herapy.grpc.rpc_pb2_grpc.AergoRPCServiceServicer method), 11
GetAccounts() (aergo.herapy.grpc.rpc_pb2_grpc.AergoRPCServiceServicer method), 12
GetAccountVotes() (aergo.herapy.grpc.rpc_pb2_grpc.AergoRPCServiceServicer method), 11
GetBlock() (aergo.herapy.grpc.rpc_pb2_grpc.AergoRPCServiceServicer method), 12
GetBlockBody() (aergo.herapy.grpc.rpc_pb2_grpc.AergoRPCServiceServicer method), 12
GetBlockMetadata() (aergo.herapy.grpc.rpc_pb2_grpc.AergoRPCServiceServicer method), 12
GetBlockTX() (aergo.herapy.grpc.rpc_pb2_grpc.AergoRPCServiceServicer method), 12
GetChainInfo() (aergo.herapy.grpc.rpc_pb2_grpc.AergoRPCServiceServicer method), 12
GetConfChangeProgress() (aergo.herapy.grpc.rpc_pb2_grpc.AergoRPCServiceServicer method), 12
GetConsensusInfo() (aergo.herapy.grpc.rpc_pb2_grpc.AergoRPCServiceServicer method), 12
GetEnterpriseConfig() (aergo.herapy.grpc.rpc_pb2_grpc.AergoRPCServiceServicer method), 12
GetNameInfo() (aergo.herapy.grpc.rpc_pb2_grpc.AergoRPCServiceServicer method), 12
GetPeers() (aergo.herapy.grpc.rpc_pb2_grpc.AergoRPCServiceServicer method), 12
GetReceipt() (aergo.herapy.grpc.rpc_pb2_grpc.AergoRPCServiceServicer method), 12
GetServerInfo() (aergo.herapy.grpc.rpc_pb2_grpc.AergoRPCServiceServicer method), 12
GetStaking() (aergo.herapy.grpc.rpc_pb2_grpc.AergoRPCServiceServicer method), 12
GetState() (aergo.herapy.grpc.rpc_pb2_grpc.AergoRPCServiceServicer method), 12
GetStateAndProof() (aergo.herapy.grpc.rpc_pb2_grpc.AergoRPCServiceServicer method), 12
GOVERNANCE (aergo.herapy.obj.transaction.TxType attribute), 23
GovernanceTxAddress (class in aergo.herapy.obj.address), 14
H
HANDSHAKING (aergo.herapy.status.peer_status.PeerStatus attribute), 25
height (aergo.herapy.obj.block.Block attribute), 17
height (aergo.herapy.obj.block.Block attribute), 17
I
id (aergo.herapy.obj.peer.Peer attribute), 21
import_account() (aergo.herapy.aergo.Aergo method), 30
import_account_from_keystore() (aergo.herapy.aergo.Aergo method), 31
import_account_from_keystore_file() (aergo.herapy.aergo.Aergo method), 31
ImportAccount() (aergo.herapy.grpc.rpc_pb2_grpc.AergoRPCServiceServicer method), 12
index (aergo.herapy.obj.event.Event attribute), 20
index_in_block (aergo.herapy.obj.transaction.Transaction attribute), 22
info (aergo.herapy.obj.name_info.NameInfo attribute), 20
info (aergo.herapy.obj.peer.Peer attribute), 21
InsufficientBalanceError, 9
is_active() (aergo.herapy.obj.stream.Stream method), 22
is_empty() (in module aergo.herapy.utils.encoding), 27
is_in_mempool (aergo.herapy.obj.transaction.Transaction attribute), 22
is_mainnet (aergo.herapy.obj.chain_id.ChainID attribute), 19
is_public (aergo.herapy.obj.chain_id.ChainID attribute), 19

```


J

`json()` (`aergo.herapy.account.Account` method), 28

`json()` (`aergo.herapy.obj.abi.Abi` method), 14

`json()` (`aergo.herapy.obj.block.Block` method), 17

`json()` (`aergo.herapy.obj.blockchain_info.BlockchainInfo` method), 18

`json()` (`aergo.herapy.obj.blockchain_status.BlockchainStatus` method), 18

`json()` (`aergo.herapy.obj.chain_id.ChainID` method), 19

`json()` (`aergo.herapy.obj.change_conf_info.ChangeConfInfo` method), 19

`json()` (`aergo.herapy.obj.consensus_info.ConsensusInfo` method), 19

`json()` (`aergo.herapy.obj.event.Event` method), 20

`json()` (`aergo.herapy.obj.name_info.NameInfo` method), 20

`json()` (`aergo.herapy.obj.node_info.NodeInfo` method), 20

`json()` (`aergo.herapy.obj.peer.Peer` method), 21

`json()` (`aergo.herapy.obj.transaction.Transaction` method), 23

`json()` (`aergo.herapy.obj.tx_result.TxResult` method), 23

L

`language` (`aergo.herapy.obj.abi.Abi` attribute), 14

`lib_hash` (`aergo.herapy.obj.consensus_info.ConsensusInfo` attribute), 19

`lib_no` (`aergo.herapy.obj.consensus_info.ConsensusInfo` attribute), 19

`ListBLEntries()` (`aergo.herapy.grpc.polarrpc_pb2_grpc.PolarisRPCServiceServicer` method), 11

`ListBlockHeaders()` (`aergo.herapy.grpc.rpc_pb2_grpc.AergoRPCServiceServicer` method), 12

`ListBlockMetadata()` (`aergo.herapy.grpc.rpc_pb2_grpc.AergoRPCServiceServicer` method), 12

`ListBlockMetadataStream()` (`aergo.herapy.grpc.rpc_pb2_grpc.AergoRPCServiceServicer` method), 13

`ListBlockStream()` (`aergo.herapy.grpc.rpc_pb2_grpc.AergoRPCServiceServicer` method), 13

`ListEvents()` (`aergo.herapy.grpc.rpc_pb2_grpc.AergoRPCServiceServicer` method), 13

`ListEventStream()` (`aergo.herapy.grpc.rpc_pb2_grpc.AergoRPCServiceServicer` method), 13

`lock_account()` (`aergo.herapy.aergo.Aergo` method), 31

`lock_account()` (`aergo.herapy.comm.Comm` method), 33

`LockAccount()` (`aergo.herapy.grpc.rpc_pb2_grpc.AergoRPCServiceServicer` method), 13

M

`magic` (`aergo.herapy.obj.chain_id.ChainID` attribute), 19

`max_block_size` (`aergo.herapy.obj.blockchain_info.BlockchainInfo` attribute), 18

`max_tokens` (`aergo.herapy.obj.blockchain_info.BlockchainInfo` attribute), 18

`members` (`aergo.herapy.obj.change_conf_info.ChangeConfInfo` attribute), 19

`mempool` (`aergo.herapy.obj.aergo_conf.AergoConfig` attribute), 15

`mempool_dumpfilepath` (`aergo.herapy.obj.aergo_conf.AergoConfig` attribute), 15

`mempool_enablefadeout` (`aergo.herapy.obj.aergo_conf.AergoConfig` attribute), 15

`mempool_fadeoutperiod` (`aergo.herapy.obj.aergo_conf.AergoConfig` attribute), 15

`mempool_showmetrics` (`aergo.herapy.obj.aergo_conf.AergoConfig` attribute), 15

`mempool_verifiers` (`aergo.herapy.obj.aergo_conf.AergoConfig` attribute), 15

`Metric()` (`aergo.herapy.grpc.polarrpc_pb2_grpc.PolarisRPCServiceServicer` method), 11

`MePolarisRPCServiceServicer` (`aergo.herapy.grpc.rpc_pb2_grpc.AergoRPCServiceServicer` method), 13

`minimum_staking` (`aergo.herapy.obj.blockchain_info.BlockchainInfo` attribute), 18

`monitor` (`aergo.herapy.obj.aergo_conf.AergoConfig` attribute), 15

`monitor_endpoint` (`aergo.herapy.obj.aergo_conf.AergoConfig` attribute), 15

`monitor_protocol` (`aergo.herapy.obj.aergo_conf.AergoConfig` attribute), 15

`name` (`aergo.herapy.obj.name_info.NameInfo` attribute), 20

`name_info` (`aergo.herapy.obj.name_info.NameInfo` attribute), 20

`new_account()` (`aergo.herapy.aergo.Aergo` method), 31

`new_account()` (`aergo.herapy.comm.Comm` method), 33

`new_account()` (`aergo.herapy.aergo.Aergo` method), 31

`new_account()` (`aergo.herapy.comm.Comm` method), 33

`new_account()` (`aergo.herapy.aergo.Aergo` method), 31

`new_account()` (`aergo.herapy.comm.Comm` method), 33

`new_account()` (`aergo.herapy.aergo.Aergo` method), 31

`new_account()` (`aergo.herapy.comm.Comm` method), 33

```

new_call_sc_tx() (aergo.herapy.aergo.Aergo p2p_nphiddenpeers
    method), 31 (aergo.herapy.obj.aergo_conf.AergoConfig
next() (aergo.herapy.obj.block_stream.BlockStream attribute), 16
    method), 18 p2p_npkey (aergo.herapy.obj.aergo_conf.AergoConfig
next() (aergo.herapy.obj.event_stream.EventStream attribute), 16
    method), 20 p2p_npmaxpeers (aergo.herapy.obj.aergo_conf.AergoConfig
next() (aergo.herapy.obj.stream.Stream method), 22 attribute), 16
NodeInfo (class in aergo.herapy.obj.node_info), 20 p2p_nppeerpool (aergo.herapy.obj.aergo_conf.AergoConfig
NodeState() (aergo.herapy.grpc.polarrpc_pb2_grpc.PolarisRPCServiceServicer attribute), 16
    method), 11 p2p_nptls (aergo.herapy.obj.aergo_conf.AergoConfig
NodeState() (aergo.herapy.grpc.rpc_pb2_grpc.AergoRPCServiceServicer attribute), 16
    method), 13 p2p_npusepolaris (aergo.herapy.obj.aergo_conf.AergoConfig
nonce (aergo.herapy.account.Account attribute), 28 attribute), 16
nonce (aergo.herapy.obj.transaction.Transaction payload (aergo.herapy.obj.transaction.Transaction at-
    attribute), 23 tribute), 23
NORMAL (aergo.herapy.obj.transaction.TxType at- payload_str (aergo.herapy.obj.transaction.Transaction
    tribute), 23 attribute), 23
num_of_tx (aergo.herapy.obj.block.Block attribute), Peer (class in aergo.herapy.obj.peer), 21
    17 PeerStatus (class in aergo.herapy.status.peer_status),
number_of_bp (aergo.herapy.obj.blockchain_info.BlockchainInfo 24
    attribute), 18

```

O

```

owner (aergo.herapy.obj.name_info.NameInfo at-
    tribute), 20

```

P

```

p2p (aergo.herapy.obj.aergo_conf.AergoConfig at- p2p_nphiddenpeers
    tribute), 15 (aergo.herapy.obj.aergo_conf.AergoConfig
p2p_logfullpeerid attribute), 16
    (aergo.herapy.obj.aergo_conf.AergoConfig
p2p_netprotocoladdr attribute), 16
    (aergo.herapy.obj.aergo_conf.AergoConfig
p2p_netprotocolport attribute), 16
    (aergo.herapy.obj.aergo_conf.AergoConfig
p2p_npaddpeers (aergo.herapy.obj.aergo_conf.AergoConfig attribute), 16
    attribute), 16
p2p_npaddpolarises (aergo.herapy.obj.aergo_conf.AergoConfig
    attribute), 16
p2p_npbindaddr (aergo.herapy.obj.aergo_conf.AergoConfig attribute), 16
    attribute), 16
p2p_npbindport (aergo.herapy.obj.aergo_conf.AergoConfig attribute), 16
    attribute), 16
p2p_npcert (aergo.herapy.obj.aergo_conf.AergoConfig attribute), 16
    attribute), 16
p2p_npdiscoverpeers (aergo.herapy.obj.aergo_conf.AergoConfig
    attribute), 16
p2p_npexposeself (aergo.herapy.obj.aergo_conf.AergoConfig
    attribute), 16

```

```

p2p_nphiddenpeers (aergo.herapy.obj.aergo_conf.AergoConfig
    attribute), 16
p2p_npkey (aergo.herapy.obj.aergo_conf.AergoConfig
    attribute), 16
p2p_npmaxpeers (aergo.herapy.obj.aergo_conf.AergoConfig
    attribute), 16
p2p_nppeerpool (aergo.herapy.obj.aergo_conf.AergoConfig
    attribute), 16
p2p_nptls (aergo.herapy.obj.aergo_conf.AergoConfig
    attribute), 16
p2p_npusepolaris (aergo.herapy.obj.aergo_conf.AergoConfig
    attribute), 16
payload (aergo.herapy.obj.transaction.Transaction at-
    tribute), 23
payload_str (aergo.herapy.obj.transaction.Transaction
    attribute), 23
Peer (class in aergo.herapy.obj.peer), 21
PeerStatus (class in aergo.herapy.status.peer_status),
    24
personal (aergo.herapy.obj.aergo_conf.AergoConfig
    attribute), 16
polaris (aergo.herapy.obj.aergo_conf.AergoConfig at-
    tribute), 16
polaris_allowprivate (aergo.herapy.obj.aergo_conf.AergoConfig
    attribute), 16
polaris_genesisfile (aergo.herapy.obj.aergo_conf.AergoConfig
    attribute), 16
PolarisRPCServiceServicer (class in
    aergo.herapy.grpc.polarrpc_pb2_grpc), 10
PolarisRPCServiceStub (class in
    aergo.herapy.grpc.polarrpc_pb2_grpc), 11
port (aergo.herapy.obj.peer.Peer attribute), 21
prev (aergo.herapy.obj.block.Block attribute), 17
private_key (aergo.herapy.account.Account at-
    tribute), 28
PrivateKey (class in aergo.herapy.obj.private_key),
    21
privkey_to_address() (in module
    aergo.herapy.utils.converter), 26
profileport (aergo.herapy.obj.aergo_conf.AergoConfig
    attribute), 16
PROPOSED (aergo.herapy.obj.change_conf_info.ChangeConfState
    attribute), 19
public_key (aergo.herapy.account.Account attribute),
    28
public_key (aergo.herapy.obj.address.Address
    attribute), 14
public_key (aergo.herapy.obj.block.Block attribute),
    17
public_key (aergo.herapy.obj.private_key.PrivateKey
    attribute), 21

```

`public_key_to_bytes()` (in module `aergo.herapy.obj.aergo_conf.AergoConfig` attribute), 16
`aergo.herapy.utils.converter`), 26

Q

`query_contract()` (`aergo.herapy.comm.Comm` method), 33
`query_contract_state()` (`aergo.herapy.comm.Comm` method), 33
`query_sc()` (`aergo.herapy.aergo.Aergo` method), 31
`query_sc_state()` (`aergo.herapy.aergo.Aergo` method), 31
`QueryContract()` (`aergo.herapy.grpc.rpc_pb2_grpc.AergoRPCServiceServicer` method), 13
`QueryContractState()` (`aergo.herapy.grpc.rpc_pb2_grpc.AergoRPCServiceServicer` method), 13

R

`RECEIPT` (`aergo.herapy.obj.tx_result.TxResultType` attribute), 23
`receipts_root_hash` (`aergo.herapy.obj.block.Block` attribute), 17
`receive_block_meta_stream()` (`aergo.herapy.aergo.Aergo` method), 31
`receive_block_meta_stream()` (`aergo.herapy.comm.Comm` method), 33
`receive_block_stream()` (`aergo.herapy.aergo.Aergo` method), 31
`receive_block_stream()` (`aergo.herapy.comm.Comm` method), 33
`receive_event_stream()` (`aergo.herapy.aergo.Aergo` method), 31
`receive_event_stream()` (`aergo.herapy.comm.Comm` method), 33
`RemoveBEntry()` (`aergo.herapy.grpc.polarrpc_pb2_grpc.PolarisRPCServiceServicer` method), 11
`rpc` (`aergo.herapy.obj.aergo_conf.AergoConfig` attribute), 16
`rpc_net_serviceaddr` (`aergo.herapy.obj.aergo_conf.AergoConfig` attribute), 16
`rpc_net_serviceport` (`aergo.herapy.obj.aergo_conf.AergoConfig` attribute), 16
`rpc_net_servicetrace` (`aergo.herapy.obj.aergo_conf.AergoConfig` attribute), 16
`rpc_nsallowcors` (`aergo.herapy.obj.aergo_conf.AergoConfig` attribute), 16
`rpc_nscacert` (`aergo.herapy.obj.aergo_conf.AergoConfig` attribute), 16
`rpc_nscert` (`aergo.herapy.obj.aergo_conf.AergoConfig` attribute), 16
`rpc_nskey` (`aergo.herapy.obj.aergo_conf.AergoConfig` attribute), 16
`rpc_nstls` (`aergo.herapy.obj.aergo_conf.AergoConfig` attribute), 16
`RUNNING` (`aergo.herapy.status.peer_status.PeerStatus` attribute), 25
`running()` (`aergo.herapy.obj.stream.Stream` method), 22

S

`SAVED` (`aergo.herapy.obj.change_conf_info.ChangeConfState` attribute), 19
`SC_CALL` (`aergo.herapy.obj.transaction.TxType` attribute), 23
`SC_DEPLOY` (`aergo.herapy.obj.transaction.TxType` attribute), 23
`SC_FEE_DELEGATION` (`aergo.herapy.obj.transaction.TxType` attribute), 23
`SC_REDEPLOY` (`aergo.herapy.obj.transaction.TxType` attribute), 23
`SCState` (class in `aergo.herapy.obj.sc_state`), 21
`SCStateVar` (class in `aergo.herapy.obj.sc_state`), 21
`send_payload()` (`aergo.herapy.aergo.Aergo` method), 31
`send_tx()` (`aergo.herapy.aergo.Aergo` method), 31
`send_tx()` (`aergo.herapy.comm.Comm` method), 33
`send_unsigned_tx()` (`aergo.herapy.aergo.Aergo` method), 31
`SendTX()` (`aergo.herapy.grpc.rpc_pb2_grpc.AergoRPCServiceServicer` method), 13
`serialize_sig()` (in module `aergo.herapy.utils.signature`), 27
`sign` (`aergo.herapy.obj.block.Block` attribute), 17
`sign` (`aergo.herapy.obj.transaction.Transaction` attribute), 23
`sign_msg()` (`aergo.herapy.obj.private_key.PrivateKey` method), 21
`sign_msg_hash()` (`aergo.herapy.account.Account` method), 28
`sign_str` (`aergo.herapy.obj.transaction.Transaction` attribute), 23
`SignTX()` (`aergo.herapy.grpc.rpc_pb2_grpc.AergoRPCServiceServicer` method), 13
`size` (`aergo.herapy.obj.block.Block` attribute), 17
`sql_recovery_point` (`aergo.herapy.account.Account` attribute), 28
`start()` (`aergo.herapy.obj.stream.Stream` method), 22
`started` (`aergo.herapy.obj.stream.Stream` attribute), 22
`STARTING` (`aergo.herapy.status.peer_status.PeerStatus` attribute), 25
`state` (`aergo.herapy.account.Account` attribute), 28

state (aergo.herapy.obj.change_conf_info.ChangeConfInfo attribute), 19
state (aergo.herapy.obj.peer.Peer attribute), 21
state_proof (aergo.herapy.account.Account attribute), 28
state_variables (aergo.herapy.obj.abi.Abi attribute), 14
status (aergo.herapy.obj.consensus_info.ConsensusInfo attribute), 19
stop () (aergo.herapy.obj.stream.Stream method), 22
stopped (aergo.herapy.obj.stream.Stream attribute), 22
STOPPED (aergo.herapy.status.peer_status.PeerStatus attribute), 25
storage_keys (aergo.herapy.obj.var_proof.VarProofs attribute), 24
storage_root (aergo.herapy.account.Account attribute), 28
Stream (class in aergo.herapy.obj.stream), 22
SUCCESS (aergo.herapy.status.tx_result_status.TxResultStatus attribute), 25
SYSTEM (aergo.herapy.obj.address.GovernanceTxAddress attribute), 14

T

timestamp (aergo.herapy.obj.block.Block attribute), 17
to_address (aergo.herapy.obj.transaction.Transaction attribute), 23
total_staking (aergo.herapy.obj.blockchain_info.BlockchainInfo attribute), 18
Transaction (class in aergo.herapy.obj.transaction), 22
TRANSFER (aergo.herapy.obj.transaction.TxType attribute), 23
transfer () (aergo.herapy.aergo.Aergo method), 31
TX_ALREADY_EXISTS (aergo.herapy.status.commit_status.CommitStatus attribute), 24
TX_HAS_SAME_NONCE (aergo.herapy.status.commit_status.CommitStatus attribute), 24
tx_hash (aergo.herapy.obj.event.Event attribute), 20
tx_hash (aergo.herapy.obj.transaction.Transaction attribute), 23
tx_index (aergo.herapy.obj.event.Event attribute), 20
TX_INSUFFICIENT_BALANCE (aergo.herapy.status.commit_status.CommitStatus attribute), 24
TX_INTERNAL_ERROR (aergo.herapy.status.commit_status.CommitStatus attribute), 24
TX_INVALID_FORMAT (aergo.herapy.status.commit_status.CommitStatus attribute), 24
TX_INVALID_HASH (aergo.herapy.status.commit_status.CommitStatus attribute), 24
TX_INVALID_SIGN (aergo.herapy.status.commit_status.CommitStatus attribute), 24
tx_list (aergo.herapy.obj.block.Block attribute), 17
TX_NONCE_TOO_LOW (aergo.herapy.status.commit_status.CommitStatus attribute), 24
TX_OK (aergo.herapy.status.commit_status.CommitStatus attribute), 24
tx_to_formatted_json () (in module aergo.herapy.utils.converter), 26
tx_to_grpc_tx () (in module aergo.herapy.utils.converter), 26
tx_to_json () (in module aergo.herapy.utils.converter), 26
tx_type (aergo.herapy.obj.transaction.Transaction attribute), 23
TxHash (class in aergo.herapy.obj.tx_hash), 23
TxResult (class in aergo.herapy.obj.tx_result), 23
TxResultStatus (class in aergo.herapy.status.tx_result_status), 25
TxResultType (class in aergo.herapy.obj.tx_result), 23
txs_root_hash (aergo.herapy.obj.block.Block attribute), 17
TxType (class in aergo.herapy.obj.transaction), 23
type (aergo.herapy.obj.consensus_info.ConsensusInfo attribute), 19
type (aergo.herapy.obj.tx_result.TxResult attribute), 23

U

uncompress_key () (in module aergo.herapy.utils.signature), 27
unlock_account () (aergo.herapy.aergo.Aergo method), 31
unlock_account () (aergo.herapy.comm.Comm method), 33
UnlockAccount () (aergo.herapy.grpc.rpc_pb2_grpc.AergoRPCService method), 13
usetestnet (aergo.herapy.obj.aergo_conf.AergoConfig attribute), 16

V

value (aergo.herapy.obj.address.Address attribute), 14
value (aergo.herapy.obj.block_hash.BlockHash attribute), 17
var_proofs (aergo.herapy.obj.sc_state.SCState attribute), 21
var_proofs (aergo.herapy.obj.var_proof.VarProofs attribute), 24
VarProofs (class in aergo.herapy.obj.var_proof), 24
verify_exclusion () (in module aergo.herapy.utils.merkle_proof), 27

`verify_exclusion_c()` (in module `aergo.herapy.utils.merkle_proof`), 27

`verify_inclusion()` (in module `aergo.herapy.utils.merkle_proof`), 27

`verify_inclusion_c()` (in module `aergo.herapy.utils.merkle_proof`), 27

`verify_proof()` (`aergo.herapy.account.Account` method), 28

`verify_proof()` (`aergo.herapy.obj.sc_state.SCState` method), 21

`verify_proof()` (`aergo.herapy.obj.var_proof.VarProofs` method), 24

`verify_proof()` (in module `aergo.herapy.utils.merkle_proof`), 27

`verify_proof_c()` (in module `aergo.herapy.utils.merkle_proof`), 27

`verify_sig()` (in module `aergo.herapy.utils.signature`), 27

`verify_sign()` (`aergo.herapy.account.Account` method), 28

`verify_sign()` (`aergo.herapy.obj.private_key.PrivateKey` method), 21

`verify_var_proof()` (`aergo.herapy.obj.var_proof.VarProofs` method), 24

`VerifyTX()` (`aergo.herapy.grpc.rpc_pb2_grpc.AergoRPCServiceServicer` method), 13

`version` (`aergo.herapy.obj.abi.Abi` attribute), 14

W

`wait_tx_result()` (`aergo.herapy.aergo.Aergo` method), 32

`WhiteList()` (`aergo.herapy.grpc.polarrrpc_pb2_grpc.PolarisRPCServiceServicer` method), 11