Safe-eth-py

Uxio

Jun 07, 2022

INTRO

Safe-eth-py includes a set of libraries to work with Ethereum and Gnosis projects:

- *EthereumClient*, a wrapper over Web3.py *Web3* client including utilities to deal with ERC20/721 tokens and tracing.
- Gnosis Safe classes and utilities.
- Price oracles for *Uniswap*, *Kyber*…
- Django serializers, models and utils.

CHAPTER

ONE

TABLE OF CONTENTS

1.1 Quick start

Just run `pip install safe-eth-py` or add it to your **requirements.txt**

If you want django ethereum utils (models, serializers, filters...) you need to run `pip install safe-eth-py[django]`

If you have issues building **coincurve** maybe you are missing some libraries

1.2 Ethereum utils

1.2.1 gnosis.eth

- `class EthereumClient (ethereum_node_url: str)`: Class to connect and do operations with a ethereum node. Uses web3 and raw rpc calls for things not supported in web3. Only `http/https` urls are suppored for the node url.

`EthereumClient` has some utils that improve a lot performance using Ethereum nodes, like the possibility of doing `batch_calls` (a single request making read-only calls to multiple contracts):

```
from gnosis.eth import EthereumClient
from gnosis.eth.contracts import get_erc721_contract
ethereum_client = EthereumClient(ETHEREUM_NODE_URL)
erc721_contract = get_erc721_contract(self.w3, token_address)
name, symbol = ethereum_client.batch_call([
                    erc721_contract.functions.name(),
                    erc721_contract.functions.symbol(),
                ])
```

More optimal in case you want to call the same function in multiple contracts

```
from gnosis.eth import EthereumClient
from gnosis.eth.contracts import get_erc20_contract
ethereum_client = EthereumClient(ETHEREUM_NODE_URL)
erc20_contract = get_erc20_contract(self.w3, token_address)
my_account = '0xD0E03B027A367fED4fd0E7834a82CD8A73E76B45'
name, symbol = ethereum_client.batch_call_same_function(
                    erc20_contract.functions.balanceOf(my_account),
```

(continues on next page)

(continued from previous page)

```
                    ['0x6810e776880C02933D47DB1b9fc05908e5386b96',
↪'0x6B175474E89094C44Da98b954EedeAC495271d0F']
                )
```

If you want to use the underlying web3.py library:

```
from gnosis.eth import EthereumClient
ethereum_client = EthereumClient(ETHEREUM_NODE_URL)
ethereum_client.w3.eth.get_block(57)
```

`EthereumClient` supports EIP1559 fees:

```
from gnosis.eth import TxSpeed
base_fee, priority_fee = ethereum_client.estimate_fee_eip1559(tx_speed=TxSpeed.NORMAL)
# If you want to convert a legacy tx to a EIP1559 one
eip1559_tx = ethereum_client.set_eip1559_fees(legacy_tx, tx_speed=TxSpeed.NORMAL)
```

You can modify timeouts (in seconds) for the RPC endpoints by setting *ETHEREUM_RPC_TIMEOUT* and *ETHEREUM_RPC_SLOW_TIMEOUT* as environment variables.

By default every RPC request will be retried *3* times. You can modify that by setting *ETHEREUM_RPC_RETRY_COUNT*.

1.2.2 gnosis.eth.constants

- `NULL_ADDRESS (0x000...0)`: Solidity `address(0)`.
- `SENTINEL_ADDRESS (0x000...1)`: Used for Gnosis Safe's linked lists (modules, owners...).
- Maximum an minimum values for *R*, *S* and *V* in ethereum signatures.

1.2.3 gnosis.eth.oracles

Price oracles for Uniswap, UniswapV2, Kyber, SushiSwap, Aave, Balancer, Curve, Mooniswap, Yearn... Example:

```
from gnosis.eth import EthereumClient
from gnosis.eth.oracles import UniswapV2Oracle
ethereum_client = EthereumClient(ETHEREUM_NODE_URL)
uniswap_oracle = UniswapV2Oracle(ethereum_client)
gno_token_mainnet_address = '0x6810e776880C02933D47DB1b9fc05908e5386b96'
weth_token_mainnet_address = '0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2'
price = uniswap_oracle.get_price(gno_token_mainnet_address, uniswap_oracle.weth_address)
```

1.2.4 gnosis.eth.utils

Contains utils for ethereum operations:

- `get_eth_address_with_key() -> Tuple[str, bytes]`: Returns a tuple of a valid public ethereum checksumed address with the private key.
- `generate_address_2(from_: Union[str, bytes], salt: Union[str, bytes], init_code: [str, bytes]) -> str`: Calculates the address of a new contract created using the new CREATE2 opcode.

1.3 Ethereum django (REST) utils

Django utils are available under `gnosis.eth.django`. You can find a set of helpers for working with Ethereum using Django and Django Rest framework.

It includes:

- **gnosis.eth.django.filters**: EthereumAddressFilter.
- **gnosis.eth.django.models**: Model fields (Ethereum address, Ethereum big integer field).
- **gnosis.eth.django.serializers**: Serializer fields (Ethereum address field, hexadecimal field).
- **gnosis.eth.django.validators**: Ethereum related validators.
- **gnosis.safe.serializers**: Serializers for Gnosis Safe (signature, transaction...).
- All the tests are written using Django Test suite.

1.4 Gnosis Products

1.4.1 Safe

On `gnosis.safe` there're classes to work with Gnosis Safe

```
from gnosis.eth import EthereumClient
from gnosis.safe import Safe
safe_address = ''  # Fill with checksummed version of a Safe address
ethereum_client = EthereumClient(ETHEREUM_NODE_URL)
safe = Safe(safe_address, ethereum_client)
safe_info = safe.retrieve_all_info()
```

To work with Multisig Transactions:

```
safe_tx = safe.build_multisig_tx(to, value, data, operation, safe_tx_gas, base_gas, gas_
↪price, gas_token,
                                 refund_receiver, signatures, safe_nonce)
safe_tx.sign(owner_1_private_key)
safe_tx.sign(owner_2_private_key)
safe_tx.call()  # Check it works
safe_tx.execute(tx_sender_private_key)
```

1.4.2 Protocol

On gnosis.protocol there're classes to work with Gnosis Protocol v2

```
import time
from gnosis.eth import EthereumNetwork
from gnosis.protocol import Order, OrderKind, GnosisProtocolAPI

account_address = ''  # Fill with checksummed version of a Gnosis Protocol user address
account_private_key = ''  # Fill with private key of a user address
gnosis_protocol_api = GnosisProtocolAPI(EthereumNetwork.RINKEBY)
print(gnosis_protocol_api.get_trades(owner=account_address))
buy_amount = gnosis_protocol_api.get_estimated_amount(base_token, quote_token, OrderKind.
↪SELL, sell_amount)
valid_to = int(time.time() + (24 * 60 * 60))  # Order valid for 1 day
order = Order(
      sellToken=base_token,
      buyToken=buyToken,
      receiver=receiver,
      sellAmount=sell_amount,
      buyAmount=buy_amount,
      validTo=valid_to,  # timestamp
      appData=ipfs_hash,  # IPFS hash for metadata
      fee_amount=0,  # If set to `0` it will be autodetected
      kind='sell',  # `sell` or `buy`
      partiallyFillable=True,  # `True` or `False`
      sellTokenBalance='erc20',  # `erc20`, `external` or `internal`
      buyTokenBalance='erc20',  # `erc20` or `internal`
  )
gnosis_protocol_api.place_order(order, account_private_key)
```

1.5 gnosis

1.5.1 gnosis package

Subpackages

gnosis.eth package

Subpackages

gnosis.eth.clients package

Submodules

gnosis.eth.clients.blockscout_client module

exception gnosis.eth.clients.blockscout_client.**BlockScoutConfigurationProblem**

Bases: *BlockscoutClientException*

class gnosis.eth.clients.blockscout_client.**BlockscoutClient**(*network: EthereumNetwork*)

Bases: object

NETWORK_WITH_URL = {<EthereumNetwork.XDAI: 100>: 'https://blockscout.com/poa/xdai/', <EthereumNetwork.MATIC: 137>: 'https://polygon-explorer-mainnet.chainstacklabs.com/', <EthereumNetwork.MUMBAI: 80001>: 'https://polygon-explorer-mumbai.chainstacklabs.com/', <EthereumNetwork.ENERGY_WEB_CHAIN: 246>: 'https://explorer.energyweb.org/', <EthereumNetwork.VOLTA: 73799>: 'https://volta-explorer.energyweb.org/', <EthereumNetwork.OLYMPUS: 333999>: 'https://explorer.polis.tech', <EthereumNetwork.BOBA_RINKEBY: 28>: 'https://blockexplorer.rinkeby.boba.network/', <EthereumNetwork.BOBA: 288>: 'https://blockexplorer.boba.network/', <EthereumNetwork.GATHER_DEVNET: 486217935>: 'https://devnet-explorer.gather.network/', <EthereumNetwork.GATHER_TESTNET: 356256156>: 'https://testnet-explorer.gather.network/', <EthereumNetwork.GATHER_MAINNET: 192837465>: 'https://explorer.gather.network/', <EthereumNetwork.METIS_TESTNET: 588>: 'https://stardust-explorer.metis.io/', <EthereumNetwork.METIS: 1088>: 'https://andromeda-explorer.metis.io/', <EthereumNetwork.FUSE_MAINNET: 122>: 'https://explorer.fuse.io/', <EthereumNetwork.VELAS_MAINNET: 106>: 'https://evmexplorer.velas.com/', <EthereumNetwork.REI_MAINNET: 47805>: 'https://scan.rei.network/', <EthereumNetwork.REI_TESTNET: 12357>: 'https://scan-test.rei.network/', <EthereumNetwork.METER: 82>: 'https://scan.meter.io/', <EthereumNetwork.METER_TESTNET: 83>: 'https://scan-warringstakes.meter.io/', <EthereumNetwork.GODWOKEN_TESTNET: 71401>: 'https://v1.betanet.gwscan.com/'}

build_url(*path: str*)

get_contract_metadata(*address: ChecksumAddress*) → Optional[*ContractMetadata*]

exception gnosis.eth.clients.blockscout_client.**BlockscoutClientException**

Bases: Exception

gnosis.eth.clients.contract_metadata module

class gnosis.eth.clients.contract_metadata.**ContractMetadata**(*name: Union[str, NoneType]*, *abi: List[Dict[str, Any]]*, *partial_match: bool*)

Bases: object

abi: List[Dict[str, Any]]

name: Optional[str]

partial_match: bool

gnosis.eth.clients.etherscan_client module

class gnosis.eth.clients.etherscan_client.**EtherscanClient**(*network: EthereumNetwork*, *api_key: Optional[str] = None*)

Bases: object

HTTP_HEADERS = {'User-Agent': 'curl/7.77.0'}

NETWORK_WITH_API_URL = {<EthereumNetwork.MAINNET: 1>: 'https://api.etherscan.io', <EthereumNetwork.RINKEBY: 4>: 'https://api-rinkeby.etherscan.io', <EthereumNetwork.ROPSTEN: 3>: 'https://api-ropsten.etherscan.io', <EthereumNetwork.GOERLI: 5>: 'https://api-goerli.etherscan.io/', <EthereumNetwork.KOVAN: 42>: 'https://api-kovan.etherscan.io/', <EthereumNetwork.BINANCE: 56>: 'https://api.bscscan.com', <EthereumNetwork.MATIC: 137>: 'https://api.polygonscan.com', <EthereumNetwork.OPTIMISTIC: 10>: 'https://api-optimistic.etherscan.io', <EthereumNetwork.ARBITRUM: 42161>: 'https://api.arbiscan.io', <EthereumNetwork.AVALANCHE: 43114>: 'https://api.snowtrace.io'}

NETWORK_WITH_URL = {<EthereumNetwork.MAINNET: 1>: 'https://etherscan.io', <EthereumNetwork.RINKEBY: 4>: 'https://rinkeby.etherscan.io', <EthereumNetwork.ROPSTEN: 3>: 'https://ropsten.etherscan.io', <EthereumNetwork.GOERLI: 5>: 'https://goerli.etherscan.io', <EthereumNetwork.KOVAN: 42>: 'https://kovan.etherscan.io', <EthereumNetwork.BINANCE: 56>: 'https://bscscan.com', <EthereumNetwork.MATIC: 137>: 'https://polygonscan.com', <EthereumNetwork.OPTIMISTIC: 10>: 'https://optimistic.etherscan.io', <EthereumNetwork.ARBITRUM: 42161>: 'https://arbiscan.io', <EthereumNetwork.AVALANCHE: 43114>: 'https://snowtrace.io'}

build_url(*path: str*)

get_contract_abi(*contract_address: str*, *retry: bool = True*)

get_contract_metadata(*contract_address: str*, *retry: bool = True*) → Optional[*ContractMetadata*]

get_contract_source_code(*contract_address: str*, *retry: bool = True*)

Get source code for a contract. Source code query also returns:

- ContractName: “”,
- CompilerVersion: “”,
- OptimizationUsed: “”,
- Runs: “”,
- ConstructorArguments: “”
- EVMVersion: “Default”,
- Library: “”,
- LicenseType: “”,
- Proxy: “0”,
- Implementation: “”,
- SwarmSource: “”

Parameters

- **contract_address** –
- **retry** – if True, try again if there's Rate Limit Error

Returns

exception gnosis.eth.clients.etherscan_client.**EtherscanClientConfigurationProblem**

Bases: Exception

exception gnosis.eth.clients.etherscan_client.**EtherscanClientException**

Bases: Exception

exception gnosis.eth.clients.etherscan_client.**EtherscanRateLimitError**

Bases: *EtherscanClientException*

gnosis.eth.clients.sourcify module

class gnosis.eth.clients.sourcify.**Sourcify**(*network: EthereumNetwork = EthereumNetwork.MAINNET*, *base_url: str = 'https://repo.sourcify.dev/'*)

Bases: object

Get contract metadata from Sourcify. Matches can be full or partial:

- Full: Both the source files as well as the meta data files were an exact match between the deployed bytecode and the published files.
- Partial: Source code compiles to the same bytecode and thus the contract behaves in the same way, but the source code can be different: Variables can have misleading names, comments can be different and especially the NatSpec comments could have been modified.

get_contract_metadata(*contract_address: str*) → Optional[*ContractMetadata*]

Module contents

gnosis.eth.contracts package

Module contents

Safe Addresses. Should be the same for every chain except for the ones with *chainId* protection. Check: https://github.com/safe-global/safe-deployments/tree/main/src/assets

GnosisSafe V1.3.0: 0xd9Db270c1B5E3Bd161E8c8503c55cEABeE709552 GnosisSafe V1.1.1: 0x34CfAC646f301356fAa8B21e94227e3583Fe3F5F GnosisSafe V1.1.0: 0xaE32496491b53841efb51829d6f886387708F99B GnosisSafe V1.0.0: 0xb6029EA3B2c51D09a50B53CA8012FeEB05bDa35A

Factories ProxyFactory V1.3.0: 0xa6B71E26C5e0845f74c812102Ca7114b6a896AB2 ProxyFactory V1.1.0: 0x50e55Af101C777bA7A1d560a774A82eF002ced9F ProxyFactory V1.0.0: 0x12302fE9c02ff50939BaAaaf415fc226C078613C

FallbackHandler CompatibilityFallBackHandler V1.3.0: 0xf48f2B2d2a534e402487b3ee7C18c33Aec0Fe5e4

Libraries CreateAndAddModules: 0x1a56aE690ab0818aF5cA349b7D21f1d7e76a3d36 MultiSend: 0xA238CBeb142c10Ef7Ad8442C6D1f9E89e07e7761

gnosis.eth.contracts.**generate_contract_fn**(*contract: Dict[str, Any]*)

Dynamically generate functions to work with the contracts

Parameters
contract –

Returns

gnosis.eth.contracts.**get_compatibility_fallback_handler_V1_3_0_contract**(*w3: Web3*, *address: Optional[ChecksumAddress] = None*)

gnosis.eth.contracts.**get_cpk_factory_contract**(*w3: Web3*, *address: Optional[ChecksumAddress] = None*)

gnosis.eth.contracts.**get_delegate_constructor_proxy_contract**(*w3: Web3*, *address: Optional[ChecksumAddress] = None*)

gnosis.eth.contracts.**get_erc1155_contract**(*w3: Web3*, *address: Optional[ChecksumAddress] = None*)

gnosis.eth.contracts.**get_erc20_contract**(*w3: Web3*, *address: Optional[ChecksumAddress] = None*)

gnosis.eth.contracts.**get_erc721_contract**(*w3: Web3*, *address: Optional[ChecksumAddress] = None*)

gnosis.eth.contracts.**get_example_erc20_contract**(*w3: Web3*, *address: Optional[ChecksumAddress] = None*)

gnosis.eth.contracts.**get_kyber_network_proxy_contract**(*w3: Web3*, *address: Optional[ChecksumAddress] = None*)

gnosis.eth.contracts.**get_multi_send_contract**(*w3: Web3*, *address: Optional[ChecksumAddress] = None*)

gnosis.eth.contracts.**get_paying_proxy_contract**(*w3: Web3*, *address: Optional[ChecksumAddress] = None*)

gnosis.eth.contracts.**get_paying_proxy_deployed_bytecode**() → bytes

gnosis.eth.contracts.**get_proxy_1_0_0_deployed_bytecode**() → bytes

gnosis.eth.contracts.**get_proxy_1_1_1_deployed_bytecode**() → bytes

gnosis.eth.contracts.**get_proxy_1_1_1_mainnet_deployed_bytecode**() → bytes

Somehow it's different from the generated version compiling the contracts

gnosis.eth.contracts.**get_proxy_1_3_0_deployed_bytecode**() → bytes

gnosis.eth.contracts.**get_proxy_contract**(*w3: Web3*, *address: Optional[ChecksumAddress] = None*)

gnosis.eth.contracts.**get_proxy_factory_V1_0_0_contract**(*w3: Web3*, *address: Optional[ChecksumAddress] = None*)

gnosis.eth.contracts.**get_proxy_factory_V1_1_1_contract**(*w3: Web3*, *address: Optional[ChecksumAddress] = None*)

gnosis.eth.contracts.**get_proxy_factory_contract**(*w3: Web3*, *address: Optional[ChecksumAddress] = None*)

gnosis.eth.contracts.**get_safe_V0_0_1_contract**(*w3: Web3*, *address: Optional[ChecksumAddress] = None*)

gnosis.eth.contracts.**get_safe_V1_0_0_contract**(*w3: Web3*, *address: Optional[ChecksumAddress] = None*)

gnosis.eth.contracts.**get_safe_V1_1_1_contract**(*w3: Web3*, *address: Optional[ChecksumAddress] = None*)

gnosis.eth.contracts.**get_safe_V1_3_0_contract**(*w3: Web3*, *address: Optional[ChecksumAddress] = None*)

gnosis.eth.contracts.**get_safe_contract**(*w3: Web3*, *address: Optional[str] = None*) → Contract

Parameters

- **w3** –
- **address** –

Returns
Latest available safe contract (v1.3.0)

gnosis.eth.contracts.**get_uniswap_exchange_contract**(*w3: Web3*, *address: Optional[ChecksumAddress] = None*)

gnosis.eth.contracts.**get_uniswap_factory_contract**(*w3: Web3*, *address: Optional[ChecksumAddress] = None*)

gnosis.eth.contracts.**get_uniswap_v2_factory_contract**(*w3: Web3*, *address: Optional[ChecksumAddress] = None*)

gnosis.eth.contracts.**get_uniswap_v2_pair_contract**(*w3: Web3*, *address: Optional[ChecksumAddress] = None*)

gnosis.eth.contracts.**get_uniswap_v2_router_contract**(*w3: Web3*, *address: Optional[ChecksumAddress] = None*)

gnosis.eth.contracts.**load_contract_interface**(*file_name*)

gnosis.eth.django package

Subpackages

Submodules

gnosis.eth.django.filters module

class gnosis.eth.django.filters.**EthereumAddressFieldForm**(**, max_length=None, min_length=None, strip=True, empty_value='', **kwargs*)

Bases: CharField

default_error_messages

prepare_value(*value*)

to_python(*value*)

Return a string.

class gnosis.eth.django.filters.**EthereumAddressFilter**(*field_name=None*, *lookup_expr=None*, ***, *label=None*, *method=None*, *distinct=False*, *exclude=False*, ***kwargs*)

Bases: Filter

field_class

alias of *EthereumAddressFieldForm*

class gnosis.eth.django.filters.**Keccak256FieldForm**(***, *max_length=None*, *min_length=None*, *strip=True*, *empty_value=''*, ***kwargs*)

Bases: CharField

default_error_messages

prepare_value(*value*)

to_python(*value*)

Return a string.

class gnosis.eth.django.filters.**Keccak256Filter**(*field_name=None*, *lookup_expr=None*, ***, *label=None*, *method=None*, *distinct=False*, *exclude=False*, ***kwargs*)

Bases: Filter

field_class

alias of *Keccak256FieldForm*

gnosis.eth.django.models module

class gnosis.eth.django.models.**EthereumAddressField**(**args*, ***kwargs*)

Bases: CharField

deconstruct()

Return enough information to recreate the field as a 4-tuple:

- The name of the field on the model, if contribute_to_class() has been run.
- The import path of the field, including the class:e.g. django.db.models.IntegerField This should be the most portable version, so less specific may be better.
- A list of positional arguments.
- A dict of keyword arguments.

Note that the positional or keyword arguments must contain values of the following types (including inner values of collection types):

- None, bool, str, int, float, complex, set, frozenset, list, tuple, dict
- UUID
- datetime.datetime (naive), datetime.date

- top-level classes, top-level functions - will be referenced by their full import path
- Storage instances - these have their own deconstruct() method

This is because the values here must be serialized into a text format (possibly new Python code, possibly JSON) and these are the only types with encoding handlers defined.

There's no need to return the exact way the field was instantiated this time, just ensure that the resulting field is the same - prefer keyword arguments over positional ones, and omit parameters with their default values.

default_error_messages

default_validators = [<function validate_checksumed_address>]

description = 'DEPRECATED. Use `EthereumAddressV2Field`. Ethereum address (EIP55)'

from_db_value(*value*, *expression*, *connection*)

get_prep_value(*value*)

Perform preliminary non-db specific value checks and conversions.

to_python(*value*)

Convert the input value into the expected Python data type, raising django.core.exceptions.ValidationError if the data can't be converted. Return the converted value. Subclasses should override this.

class gnosis.eth.django.models.**EthereumAddressV2Field**(*verbose_name=None*, *name=None*, *primary_key=False*, *max_length=None*, *unique=False*, *blank=False*, *null=False*, *db_index=False*, *rel=None*, *default=<class 'django.db.models.fields.NOT_PROVIDED'>*, *editable=True*, *serialize=True*, *unique_for_date=None*, *unique_for_month=None*, *unique_for_year=None*, *choices=None*, *help_text=''*, *db_column=None*, *db_tablespace=None*, *auto_created=False*, *validators=()*, *error_messages=None*)

Bases: Field

default_error_messages

default_validators = [<function validate_checksumed_address>]

description = 'Ethereum address (EIP55)'

formfield(***kwargs*)

Return a django.forms.Field instance for this field.

from_db_value(*value: memoryview*, *expression*, *connection*) → Optional[ChecksumAddress]

get_internal_type()

get_prep_value(*value: ChecksumAddress*) → Optional[bytes]

Perform preliminary non-db specific value checks and conversions.

to_python(*value*) → Optional[ChecksumAddress]

Convert the input value into the expected Python data type, raising django.core.exceptions.ValidationError if the data can't be converted. Return the converted value. Subclasses should override this.

class gnosis.eth.django.models.**HexField**(**args*, *db_collation=None*, ***kwargs*)

Bases: CharField

Field to store hex values (without 0x). Returns hex with 0x prefix.

On Database side a CharField is used.

clean(*value*, *model_instance*)

Convert the value's type and run validation. Validation errors from to_python() and validate() are propagated. Return the correct value if no error is raised.

description = 'Stores a hex value into an CharField'

formfield(***kwargs*)

Return a django.forms.Field instance for this field.

from_db_value(*value*, *expression*, *connection*)

get_prep_value(*value*)

Perform preliminary non-db specific value checks and conversions.

to_python(*value*)

Convert the input value into the expected Python data type, raising django.core.exceptions.ValidationError if the data can't be converted. Return the converted value. Subclasses should override this.

class gnosis.eth.django.models.**Keccak256Field**(**args*, ***kwargs*)

Bases: BinaryField

default_error_messages

description = 'Keccak256 hash stored as binary'

formfield(***kwargs*)

Return a django.forms.Field instance for this field.

from_db_value(*value: memoryview*, *expression*, *connection*) → Optional[bytes]

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

Perform preliminary non-db specific value checks and conversions.

to_python(*value*) → Optional[str]

Convert the input value into the expected Python data type, raising django.core.exceptions.ValidationError if the data can't be converted. Return the converted value. Subclasses should override this.

class gnosis.eth.django.models.**Sha3HashField**(**args*, ***kwargs*)

Bases: *HexField*

deconstruct()

Return enough information to recreate the field as a 4-tuple:

- The name of the field on the model, if contribute_to_class() has been run.
- The import path of the field, including the class:e.g. django.db.models.IntegerField This should be the most portable version, so less specific may be better.
- A list of positional arguments.
- A dict of keyword arguments.

Note that the positional or keyword arguments must contain values of the following types (including inner values of collection types):

- None, bool, str, int, float, complex, set, frozenset, list, tuple, dict
- UUID
- datetime.datetime (naive), datetime.date
- top-level classes, top-level functions - will be referenced by their full import path
- Storage instances - these have their own deconstruct() method

This is because the values here must be serialized into a text format (possibly new Python code, possibly JSON) and these are the only types with encoding handlers defined.

There's no need to return the exact way the field was instantiated this time, just ensure that the resulting field is the same - prefer keyword arguments over positional ones, and omit parameters with their default values.

description = 'DEPRECATED. Use `Keccak256Field`'

class gnosis.eth.django.models.**Uint256Field**(**args*, ***kwargs*)

Bases: DecimalField

deconstruct()

Return enough information to recreate the field as a 4-tuple:

- The name of the field on the model, if contribute_to_class() has been run.
- The import path of the field, including the class:e.g. django.db.models.IntegerField This should be the most portable version, so less specific may be better.
- A list of positional arguments.
- A dict of keyword arguments.

Note that the positional or keyword arguments must contain values of the following types (including inner values of collection types):

- None, bool, str, int, float, complex, set, frozenset, list, tuple, dict
- UUID
- datetime.datetime (naive), datetime.date
- top-level classes, top-level functions - will be referenced by their full import path
- Storage instances - these have their own deconstruct() method

This is because the values here must be serialized into a text format (possibly new Python code, possibly JSON) and these are the only types with encoding handlers defined.

There's no need to return the exact way the field was instantiated this time, just ensure that the resulting field is the same - prefer keyword arguments over positional ones, and omit parameters with their default values.

description

Field to store ethereum uint256 values. Uses Decimal db type without decimals to store in the database, but retrieve as *int* instead of *Decimal* (https://docs.python.org/3/library/decimal.html)

from_db_value(*value*, *expression*, *connection*)

gnosis.eth.django.serializers module

class gnosis.eth.django.serializers.**EthereumAddressField**(**args*, ***kwargs*)

Bases: Field

Ethereum address checksumed https://github.com/ethereum/EIPs/blob/master/EIPS/eip-55.md

to_internal_value(*data*)

Transform the *incoming* primitive data into a native value.

to_representation(*obj*)

Transform the *outgoing* native value into primitive data.

class gnosis.eth.django.serializers.**HexadecimalField**(**args*, ***kwargs*)

Bases: Field

Serializes hexadecimal values starting by *0x*. Empty values should be None or just *0x*.

default_error_messages

to_internal_value(*data*)

Transform the *incoming* primitive data into a native value.

to_representation(*obj*)

Transform the *outgoing* native value into primitive data.

class gnosis.eth.django.serializers.**Sha3HashField**(**args*, ***kwargs*)

Bases: *HexadecimalField*

class gnosis.eth.django.serializers.**SignatureSerializer**(**args*, ***kwargs*)

Bases: Serializer

class gnosis.eth.django.serializers.**TransactionResponseSerializer**(**args*, ***kwargs*)

Bases: Serializer

Use chars to avoid problems with big ints (i.e. JavaScript)

get_fields()

Returns a dictionary of {field_name: field_instance}.

class gnosis.eth.django.serializers.**TransactionSerializer**(**args*, ***kwargs*)

Bases: Serializer

get_fields()

Returns a dictionary of {field_name: field_instance}.

gnosis.eth.django.validators module

gnosis.eth.django.validators.**validate_checksumed_address**(*address*)

Module contents

gnosis.eth.oracles package

Subpackages

gnosis.eth.oracles.abis package

Submodules

gnosis.eth.oracles.abis.aave_abis module

gnosis.eth.oracles.abis.balancer_abis module

gnosis.eth.oracles.abis.curve_abis module

gnosis.eth.oracles.abis.mooniswap_abis module

gnosis.eth.oracles.abis.yearn_abis module

Module contents

Submodules

gnosis.eth.oracles.oracles module

class gnosis.eth.oracles.oracles.**AaveOracle**(*ethereum_client:* EthereumClient, *price_oracle:* PriceOracle)

Bases: *PriceOracle*

get_price(*token_address: str*) → float

class gnosis.eth.oracles.oracles.**BalancerOracle**(*ethereum_client:* EthereumClient, *price_oracle:* PriceOracle)

Bases: *PricePoolOracle*

Oracle for Balancer. More info on https://balancer.exchange

get_pool_token_price(*pool_token_address: ChecksumAddress*) → float

Estimate balancer pool token price based on its components

Parameters
pool_token_address – Balancer pool token address

Returns
Eth price for pool token

Raises
CannotGetPriceFromOracle

exception gnosis.eth.oracles.oracles.**CannotGetPriceFromOracle**

Bases: *OracleException*

class gnosis.eth.oracles.oracles.**ComposedPriceOracle**

Bases: ABC

abstract get_underlying_tokens(**args*) → List[Tuple[*UnderlyingToken*]]

class gnosis.eth.oracles.oracles.**CreamOracle**(*ethereum_client:* EthereumClient, *price_oracle:* PriceOracle)

Bases: *PriceOracle*

get_price(*token_address: str*) → float

class gnosis.eth.oracles.oracles.**CurveOracle**(*ethereum_client:* EthereumClient, *zerion_adapter_address: Optional[str] = None*)

Bases: *ZerionComposedOracle*

Curve pool Oracle. More info on https://curve.fi/

ZERION_ADAPTER_ADDRESS = '0x99b0bEadc3984eab9842AF81f9fad0C2219108cc'

get_underlying_tokens(*token_address: ChecksumAddress*) → List[*UnderlyingToken*]

Check if passed token address is a Curve gauge deposit token, if it's a gauge we replace the address with the corresponding LP token address More info on https://resources.curve.fi/base-features/understanding-gauges

class gnosis.eth.oracles.oracles.**EnzymeOracle**(*ethereum_client:* EthereumClient, *zerion_adapter_address: Optional[str] = None*)

Bases: *ZerionComposedOracle*

Enzyme pool Oracle. More info on https://enzyme.finance/

ZERION_ADAPTER_ADDRESS = '0x9e71455D748C23566b19493D09435574097C7D67'

exception gnosis.eth.oracles.oracles.**InvalidPriceFromOracle**

Bases: *OracleException*

class gnosis.eth.oracles.oracles.**KyberOracle**(*ethereum_client:* EthereumClient, *kyber_network_proxy_address: Optional[str] = None*)

Bases: *PriceOracle*

ADDRESSES = {<EthereumNetwork.MAINNET: 1>:
'0x9AAb3f75489902f3a48495025729a0AF77d4b11e', <EthereumNetwork.RINKEBY: 4>:
'0x0d5371e5EE23dec7DF251A8957279629aa79E9C5', <EthereumNetwork.ROPSTEN: 3>:
'0xd719c34261e099Fdb33030ac8909d5788D3039C4', <EthereumNetwork.KOVAN: 42>:
'0xc153eeAD19e0DBbDb3462Dcc2B703cC6D738A37c'}

ETH_TOKEN_ADDRESS = '0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE'

get_price(*token_address_1: str*, *token_address_2: str = '0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE'*) → float

property kyber_network_proxy_address

property kyber_network_proxy_contract

class gnosis.eth.oracles.oracles.**MooniswapOracle**(*ethereum_client:* EthereumClient, *price_oracle:* PriceOracle)

Bases: *BalancerOracle*

get_pool_token_price(*pool_token_address: ChecksumAddress*) → float

Estimate balancer pool token price based on its components

Parameters
pool_token_address – Moniswap pool token address

Returns
Eth price for pool token

Raises
CannotGetPriceFromOracle

exception gnosis.eth.oracles.oracles.**OracleException**

Bases: Exception

class gnosis.eth.oracles.oracles.**PoolTogetherOracle**(*ethereum_client: EthereumClient*, *zerion_adapter_address: Optional[str] = None*)

Bases: *ZerionComposedOracle*

PoolTogether pool Oracle. More info on https://pooltogether.com/

ZERION_ADAPTER_ADDRESS = '0xb4E0E1672fFd9b128784dB9f3BE9158fac3f1DFc'

class gnosis.eth.oracles.oracles.**PriceOracle**

Bases: ABC

abstract get_price(**args*) → float

class gnosis.eth.oracles.oracles.**PricePoolOracle**

Bases: ABC

abstract get_pool_token_price(*pool_token_address: ChecksumAddress*) → float

class gnosis.eth.oracles.oracles.**SushiswapOracle**(*ethereum_client: EthereumClient*, *router_address: Optional[str] = None*)

Bases: *UniswapV2Oracle*

pair_init_code =
HexBytes('0xe18a34eb0e04b04f7a0ac29a6e80748dca96319b42c54d679cb821dca90c6303')

router_address: str = '0xd9e1cE17f2641f24aE83637ab66a2cca9C378B9F'

class gnosis.eth.oracles.oracles.**UnderlyingToken**(*address: <function NewType.<locals>.new_type at 0x7fe1df3cab80>*, *quantity: int*)

Bases: object

address: ChecksumAddress

quantity: int

class gnosis.eth.oracles.oracles.**UniswapOracle**(*ethereum_client: EthereumClient*, *uniswap_factory_address: Optional[str] = None*)

Bases: *PriceOracle*

ADDRESSES = {<EthereumNetwork.MAINNET: 1>:
'0xc0a47dFe034B400B47bDaD5FecDa2621de6c4d95', <EthereumNetwork.RINKEBY: 4>:
'0xf5D915570BC477f9B8D6C0E980aA81757A3AaC36', <EthereumNetwork.ROPSTEN: 3>:
'0x9c83dCE8CA20E9aAF9D3efc003b2ea62aBC08351', <EthereumNetwork.KOVAN: 42>:
'0xD3E51Ef092B2845f10401a0159B2B96e8B6c3D30', <EthereumNetwork.GOERLI: 5>:
'0x6Ce570d02D73d4c384b46135E87f8C592A8c86dA'}

get_price(*token_address: str*) → float

get_uniswap_exchange(*token_address: str*) → str

property uniswap_factory

property uniswap_factory_address

class gnosis.eth.oracles.oracles.**UniswapV2Oracle**(*ethereum_client:* EthereumClient, *router_address: Optional[str] = None*)

Bases: *PricePoolOracle*, *PriceOracle*

calculate_pair_address(*token_address: str*, *token_address_2: str*)

Calculate pair address without querying blockchain. https://uniswap.org/docs/v2/smart-contract-integration/getting-pair-addresses/#docs-header

Parameters

- **token_address** –
- **token_address_2** –

Returns
Checksummed address for token pair. It could be not created yet

property factory

property factory_address: str

Returns
Uniswap factory checksummed address

Raises
BadFunctionCallOutput: If router contract is not deployed

get_decimals(*token_address: str*, *token_address_2: str*) → Tuple[int, int]

get_pair_address(*token_address: str*, *token_address_2: str*) → Optional[str]

Get uniswap pair address. *token_address* and *token_address_2* are interchangeable. https://uniswap.org/docs/v2/smart-contracts/factory/

Parameters

- **token_address** –
- **token_address_2** –

Returns
Address of the pair for *token_address* and *token_address_2*, if it has been created, else *None*.

get_pool_token_price(*pool_token_address: ChecksumAddress*) → float

Estimate pool token price based on its components

Parameters
pool_token_address –

Returns
Pool token eth price per unit (total pool token supply / 1e18)

Raises
CannotGetPriceFromOracle

get_price(*token_address: str*, *token_address_2: Optional[str] = None*) → float

get_price_without_exception(*token_address: str*, *token_address_2: Optional[str] = None*) → float

Parameters

- **token_address** –
- **token_address_2** –

Returns
Call *get_price*, return 0. instead on an exception if there's any issue

get_reserves(*pair_address: str*) → Tuple[int, int]

Returns the Also returns the block.timestamp (mod 2**32) of the last block during which an interaction occured for the pair. https://uniswap.org/docs/v2/smart-contracts/pair/ :return: Reserves of *token_address* and *token_address_2* used to price trades and distribute liquidity.

pair_init_code = HexBytes('0x96e8ac4277198ff8b6f785478aa9a39f403cb768dd02cbee326c3e7da348845f')

router_address: str = '0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D'

property weth_address: str

Returns
Wrapped ether checksummed address

Raises
BadFunctionCallOutput: If router contract is not deployed

class gnosis.eth.oracles.oracles.**UsdPricePoolOracle**

Bases: ABC

abstract get_pool_usd_token_price(*pool_token_address: ChecksumAddress*) → float

class gnosis.eth.oracles.oracles.**YearnOracle**(*ethereum_client: EthereumClient*, *yearn_vault_token_adapter: Optional[str] = '0xb460FcC1B6c1CBD7D03F47B6BD5F03994d286c75'*, *iearn_token_adapter: Optional[str] = '0x65B23774daE2a5be02dD275918DDF048d177a5B4'*)

Bases: *ComposedPriceOracle*

Yearn oracle. More info on https://docs.yearn.finance

get_underlying_tokens(*token_address: ChecksumAddress*) → List[Tuple[float, ChecksumAddress]]

Parameters
token_address –

Returns
Price per share and underlying token

Raises
CannotGetPriceFromOracle

class gnosis.eth.oracles.oracles.**ZerionComposedOracle**(*ethereum_client: EthereumClient*, *zerion_adapter_address: Optional[str] = None*)

Bases: *ComposedPriceOracle*

ZERION_ADAPTER_ADDRESS = None

get_underlying_tokens(*token_address: ChecksumAddress*) → List[*UnderlyingToken*]

Use Zerion Token adapter to return underlying components for pool

Parameters
token_address – Pool token address

Returns
Price per share and underlying token

Raises
CannotGetPriceFromOracle

property zerion_adapter_contract: Optional[Contract]

Returns
https://curve.readthedocs.io/registry-registry.html

Module contents

Submodules

gnosis.eth.constants module

gnosis.eth.ethereum_client module

class gnosis.eth.ethereum_client.**BatchCallManager**(*ethereum_client:* EthereumClient)

Bases: *EthereumClientManager*

batch_call(*contract_functions: Iterable[ContractFunction]*, *from_address: Optional[ChecksumAddress] = None*, *raise_exception: bool = True*, *block_identifier: Optional[Union[Literal['latest', 'earliest', 'pending'], BlockNumber, Hash32, HexStr, HexBytes, int]] = 'latest'*) → List[Optional[Any]]

Do batch requests of multiple contract calls

Parameters

- **contract_functions** – Iterable of contract functions using web3.py contracts. For instance, a valid argument would be [erc20_contract.functions.balanceOf(address), erc20_contract.functions.decimals()]
- **from_address** – Use this address as *from* in every call if provided
- **block_identifier** – *latest* by default
- **raise_exception** – If False, exception will not be raised if there's any problem and instead *None* will be returned as the value.

Returns
List with the ABI decoded return values

batch_call_custom(*payloads: Iterable[Dict[str, Any]]*, *raise_exception: bool = True*, *block_identifier: Optional[Union[Literal['latest', 'earliest', 'pending'], BlockNumber, Hash32, HexStr, HexBytes, int]] = 'latest'*) → List[Optional[Any]]

Do batch requests of multiple contract calls

Parameters

- **payloads** – Iterable of Dictionaries with at least {'data': '<hex-string>', 'output_type': <solidity-output-type>, 'to': '<checksummed-address>'}. *from* can also be provided and if *fn_name* is provided it will be used for debugging purposes
- **raise_exception** – If False, exception will not be raised if there's any problem and instead *None* will be returned as the value
- **block_identifier** – *latest* by default

Returns
List with the ABI decoded return values

Raises
ValueError if raise_exception=True

batch_call_same_function(*contract_function: ContractFunction*, *contract_addresses: Sequence[ChecksumAddress]*, *from_address: Optional[ChecksumAddress] = None*, *raise_exception: bool = True*, *block_identifier: Optional[Union[Literal['latest', 'earliest', 'pending'], BlockNumber, Hash32, HexStr, HexBytes, int]] = 'latest'*) → List[Optional[Any]]

Do batch requests using the same function to multiple address. `batch_call` could be used to achieve that, but generating the ContractFunction is slow, so this function allows to use the same contract_function for multiple addresses

Parameters

- **contract_function** –
- **contract_addresses** –
- **from_address** –
- **raise_exception** –
- **block_identifier** –

Returns

class gnosis.eth.ethereum_client.**Erc20Info**(*name*, *symbol*, *decimals*)

Bases: `tuple`

decimals: int
Alias for field number 2

name: str
Alias for field number 0

symbol: str
Alias for field number 1

class gnosis.eth.ethereum_client.**Erc20Manager**(*ethereum_client:* EthereumClient)

Bases: *EthereumClientManager*

Manager for ERC20 operations

TRANSFER_TOPIC = HexBytes('0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef')

decode_logs(*logs: List[LogReceipt]*)

get_balance(*address: str*, *token_address: str*) → int

Get balance of address for *erc20_address*

Parameters

- **address** – owner address
- **token_address** – erc20 token address

Returns
balance

get_balances(*address: str*, *token_addresses: List[str]*) → List[*BalanceDict*]

Get balances for Ether and tokens for an *address*

Parameters

- **address** – Owner address checksummed
- **token_addresses** – token addresses to check

Returns
`List[BalanceDict]`

get_decimals(*erc20_address: str*) → int

get_info(*erc20_address: str*) → *Erc20Info*

Get erc20 information (*name*, *symbol* and *decimals*). Use batching to get all info in the same request.

Parameters
erc20_address –

Returns
Erc20Info

Raises
InvalidERC20Info

get_name(*erc20_address: str*) → str

get_symbol(*erc20_address: str*) → str

get_total_transfer_history(*addresses: Optional[Sequence[ChecksumAddress]] = None*, *from_block: Union[Literal['latest', 'earliest', 'pending'], BlockNumber, Hash32, HexStr, HexBytes, int] = 0*, *to_block: Optional[Union[Literal['latest', 'earliest', 'pending'], BlockNumber, Hash32, HexStr, HexBytes, int]] = None*, *token_address: Optional[ChecksumAddress] = None*) → List[Dict[str, Any]]

Get events for erc20 and erc721 transfers from and to an *address*. We decode it manually. Example of an erc20 event:

```
{'logIndex': 0,
 'transactionIndex': 0,
 'transactionHash': HexBytes(
↪'0x4d0f25313603e554e3b040667f7f391982babbd195c7ae57a8c84048189f7794'),
 'blockHash': HexBytes(
↪'0x90fa67d848a0eaf3be625235dae28815389f5292d4465c48d1139f0c207f8d42'),
 'blockNumber': 791,
 'address': '0xf7d0Bd47BF3214494E7F5B40E392A25cb4788620',
 'data': '0x0000000000000000000000000000000000000000000000002001f716742000',
```

(continues on next page)

(continued from previous page)

```
 'topics': [HexBytes(
↪'0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef'),
  HexBytes('0x000000000000000000000000f5984365fca2e3bc7d2e020abb2c701df9070eb7
↪'),
  HexBytes('0x000000000000000000000000000000001df62f291b2e969fb0849d99d9ce41e2f137006e
↪')],
 'type': 'mined'
 'args': {'from': '0xf5984365FcA2e3bc7D2E020AbB2c701DF9070eB7',
          'to': '0x1dF62f291b2E969fB0849d99D9Ce41e2F137006e',
          'value': 90093600000000000
         }
}
An example of an erc721 event
{'address': '0x6631FcbB50677DfC6c02CCDcc03a8f68Db427a64',
 'blockHash': HexBytes(
↪'0x95c71c6c9373e9a8ca2c767dda1cd5083eb6addcce36fc216c9e1f458d6970f9'),
 'blockNumber': 5341681,
 'data': '0x',
 'logIndex': 0,
 'removed': False,
 'topics': [HexBytes(
↪'0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef'),
  HexBytes('0x0000000000000000000000000000000000000000000000000000000000000000
↪'),
  HexBytes('0x000000000000000000000000b5239c032ab9fb5abfc3903e770a4b6a9095542c
↪'),
  HexBytes('0x0000000000000000000000000000000000000000000000000000000000000063
↪')],
 'transactionHash': HexBytes(
↪'0xce8c8af0503e6f8a421345c10cdf92834c95186916a3f5b1437d2bba63d2db9e'),
 'transactionIndex': 0,
 'transactionLogIndex': '0x0',
 'type': 'mined',
 'args': {'from': '0x0000000000000000000000000000000000000000',
          'to': '0xb5239C032AB9fB5aBFc3903e770A4B6a9095542C',
          'tokenId': 99
         }
 }
An example of unknown transfer event (no indexed parts), could be a ERC20 or␣
↪ERC721 transfer:
{'address': '0x6631FcbB50677DfC6c02CCDcc03a8f68Db427a64',
 'blockHash': HexBytes(
↪'0x95c71c6c9373e9a8ca2c767dda1cd5083eb6addcce36fc216c9e1f458d6970f9'),
 'blockNumber': 5341681,
 'data': '0x',
 'logIndex': 0,
 'removed': False,
 'topics': [HexBytes(
↪'0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef'),
  HexBytes('0x0000000000000000000000000000000000000000000000000000000000000000
↪'),
  HexBytes('0x000000000000000000000000b5239c032ab9fb5abfc3903e770a4b6a9095542c
↪'),
```

(continues on next page)

(continued from previous page)

```
  HexBytes('0x0000000000000000000000000000000000000000000000000000000000000063
↪')],
 'transactionHash': HexBytes(
↪'0xce8c8af0503e6f8a421345c10cdf92834c95186916a3f5b1437d2bba63d2db9e'),
 'transactionIndex': 0,
 'transactionLogIndex': '0x0',
 'type': 'mined',
 'args': {'from': '0x0000000000000000000000000000000000000000',
          'to': '0xb5239C032AB9fB5aBFc3903e770A4B6a9095542C',
          'unknown': 99
         }
 }
```

Parameters

- **addresses** – Search events *from* and *to* these *addresses*. If not, every transfer event within the range will be retrieved
- **from_block** – Block to start querying from
- **to_block** – Block to stop querying from
- **token_address** – Address of the token

Returns

List of events sorted by blockNumber

get_transfer_history(*from_block: int*, *to_block: Optional[int] = None*, *from_address: Optional[str] = None*, *to_address: Optional[str] = None*, *token_address: Optional[str] = None*) → List[Dict[str, Any]]

DON'T USE, it will fail in some cases until they fix https://github.com/ethereum/web3.py/issues/1351 Get events for erc20/erc721 transfers. At least one of *from_address*, *to_address* or *token_address* must be defined. Example of decoded event:

```
{
    "args": {
        "from": "0x1Ce67Ea59377A163D47DFFc9BaAB99423BE6EcF1",
        "to": "0xaE9E15896fd32E59C7d89ce7a95a9352D6ebD70E",
        "value": 15000000000000000
    },
    "event": "Transfer",
    "logIndex": 42,
    "transactionIndex": 60,
    "transactionHash":
↪"0x71d6d83fef3347bad848e83dfa0ab28296e2953de946ee152ea81c6dfb42d2b3",
    "address": "0xfecA834E7da9D437645b474450688DA9327112a5",
    "blockHash":
↪"0x054de9a496fc7d10303068cbc7ee3e25181a3b26640497859a5e49f0342e7db2",
    "blockNumber": 7265022
}
```

Parameters

- **from_block** – Block to start querying from

- **to_block** – Block to stop querying from
- **from_address** – Address sending the erc20 transfer
- **to_address** – Address receiving the erc20 transfer
- **token_address** – Address of the token

Returns
List of events (decoded)

Throws
ReadTimeout

send_tokens(*to: str*, *amount: int*, *erc20_address: str*, *private_key: str*, *nonce: Optional[int] = None*, *gas_price: Optional[int] = None*, *gas: Optional[int] = None*) → bytes

Send tokens to address

Parameters

- **to** –
- **amount** –
- **erc20_address** –
- **private_key** –
- **nonce** –
- **gas_price** –
- **gas** –

Returns
tx_hash

class gnosis.eth.ethereum_client.**Erc721Info**(*name*, *symbol*)

Bases: `tuple`

name: str

Alias for field number 0

symbol: str

Alias for field number 1

class gnosis.eth.ethereum_client.**Erc721Manager**(*ethereum_client:* EthereumClient)

Bases: *EthereumClientManager*

TRANSFER_TOPIC = HexBytes('0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef')

get_balance(*address: str*, *token_address: str*) → int

Get balance of address for *erc20_address*

Parameters

- **address** – owner address
- **token_address** – erc721 token address

Returns
balance

get_balances(*address: str*, *token_addresses: List[str]*) → List[*TokenBalance*]

Get balances for tokens for an *address*. If there's a problem with a token_address *0* will be returned for balance

Parameters

- **address** – Owner address checksummed
- **token_addresses** – token addresses to check

Returns

get_info(*token_address: str*) → *Erc721Info*

Get erc721 information (*name*, *symbol*). Use batching to get all info in the same request.

Parameters
token_address –

Returns
Erc721Info

get_owners(*token_addresses_with_token_ids: Sequence[Tuple[str, int]]*) →
List[Optional[ChecksumAddress]]

Parameters
token_addresses_with_token_ids – Tuple(token_address: str, token_id: int)

Returns
List of owner addresses, *None* if not found

get_token_uris(*token_addresses_with_token_ids: Sequence[Tuple[str, int]]*) → List[Optional[str]]

Parameters
token_addresses_with_token_ids – Tuple(token_address: str, token_id: int)

Returns
List of token_uris, *None* if not found

class gnosis.eth.ethereum_client.**EthereumClient**(*ethereum_node_url: URI = 'http://localhost:8545'*, *provider_timeout: int = 15*, *slow_provider_timeout: int = 60*, *retry_count: int = 3*, *use_caching_middleware: bool = True*)

Bases: object

Manage ethereum operations. Uses web3 for the most part, but some other stuff is implemented from scratch. Note: If you want to use *pending* state with *Parity*, it must be run with *–pruning=archive* or *–force-sealing*

NULL_ADDRESS = '0x0000000000000000000000000000000000000000'

batch_call(*contract_functions: Iterable[ContractFunction]*, *from_address: Optional[ChecksumAddress] = None*, *raise_exception: bool = True*, *force_batch_call: bool = False*, *block_identifier: Optional[Union[Literal['latest', 'earliest', 'pending'], BlockNumber, Hash32, HexStr, HexBytes, int]] = 'latest'*) → List[Optional[Union[bytes, Any]]]

Call multiple functions. Multicall contract by MakerDAO will be used by default if available

Parameters

- **contract_functions** –
- **from_address** – Only available when Multicall is not used
- **raise_exception** – If True, raise BatchCallException if one of the calls fails

- **force_batch_call** – If `True`, ignore multicall and always use batch calls to get the result (less optimal). If `False`, more optimal way will be tried.
- **block_identifier** –

Returns
List of elements decoded to their types, `None` if they cannot be decoded and bytes if a revert error is returned and `raise_exception=False`

Raises
BatchCallException

batch_call_same_function(*contract_function: ContractFunction*, *contract_addresses: Sequence[ChecksumAddress]*, *from_address: Optional[ChecksumAddress] = None*, *raise_exception: bool = True*, *force_batch_call: bool = False*, *block_identifier: Optional[Union[Literal['latest', 'earliest', 'pending'], BlockNumber, Hash32, HexStr, HexBytes, int]] = 'latest'*) → List[Optional[Union[bytes, Any]]]

Call the same function in multiple contracts. Way more optimal than using *batch_call* generating multiple `ContractFunction` objects.

Parameters

- **contract_function** –
- **contract_addresses** –
- **from_address** – Only available when `Multicall` is not used
- **raise_exception** – If `True`, raise `BatchCallException` if one of the calls fails
- **force_batch_call** – If `True`, ignore multicall and always use batch calls to get the result (less optimal). If `False`, more optimal way will be tried.
- **block_identifier** –

Returns
List of elements decoded to the same type, `None` if they cannot be decoded and bytes if a revert error is returned and `raise_exception=False`

Raises
BatchCallException

check_tx_with_confirmations(*tx_hash: Union[Hash32, HexBytes, HexStr]*, *confirmations: int*) → bool

Check tx hash and make sure it has the confirmations required

Parameters

- **tx_hash** – Hash of the tx
- **confirmations** – Minimum number of confirmations required

Returns
True if tx was mined with the number of confirmations required, False otherwise

property current_block_number

deploy_and_initialize_contract(*deployer_account: LocalAccount*, *constructor_data: bytes*, *initializer_data: bytes = b''*, *check_receipt: bool = True*)

static estimate_data_gas(*data: bytes*)

estimate_fee_eip1559(*tx_speed:* TxSpeed = *TxSpeed.NORMAL*) → Tuple[int, int]

Check https://github.com/ethereum/execution-apis/blob/main/src/eth/fee_market.json#L15

Returns
Tuple[BaseFeePerGas, MaxPriorityFeePerGas]

Raises
ValueError if not supported on the network

estimate_gas(*to: str*, *from_: Optional[str] = None*, *value: Optional[int] = None*, *data: Optional[Union[bytes, HexStr]] = None*, *gas: Optional[int] = None*, *gas_price: Optional[int] = None*, *block_identifier: Optional[Union[Literal['latest', 'earliest', 'pending'], BlockNumber, Hash32, HexStr, HexBytes, int]] = None*) → int

Estimate gas calling *eth_estimateGas*

Parameters

- **from** –
- **to** –
- **value** –
- **data** –
- **gas** –
- **gas_price** –
- **block_identifier** – Be careful, *Geth* does not support *pending* when estimating

Returns
Amount of gas needed for transaction

Raises
ValueError

get_balance(*address: ChecksumAddress*, *block_identifier: Optional[Union[Literal['latest', 'earliest', 'pending'], BlockNumber, Hash32, HexStr, HexBytes, int]] = None*)

get_block(*block_identifier: Union[Literal['latest', 'earliest', 'pending'], BlockNumber, Hash32, HexStr, HexBytes, int]*, *full_transactions: bool = False*) → Optional[BlockData]

get_blocks(*block_identifiers: Iterable[Union[Literal['latest', 'earliest', 'pending'], BlockNumber, Hash32, HexStr, HexBytes, int]]*, *full_transactions: bool = False*) → List[Optional[BlockData]]

get_chain_id() → int

Returns
ChainId returned by the RPC *eth_chainId* method. It should never change, so it's cached.

get_client_version() → str

Returns
RPC version information

get_network() → EthereumNetwork

Get network name based on the chainId

Returns
EthereumNetwork based on the chainId. If network is not on our list, *EthereumNetwork.UNKOWN* is returned

get_nonce_for_account(*address: ChecksumAddress*, *block_identifier: Optional[Union[Literal['latest', 'earliest', 'pending'], BlockNumber, Hash32, HexStr, HexBytes, int]] = 'latest'*)

Get nonce for account. *getTransactionCount* is the only method for what *pending* is currently working (Geth and Parity)

Parameters

- **address** –
- **block_identifier** –

Returns

get_transaction(*tx_hash: Union[Hash32, HexBytes, HexStr]*) → Optional[TxData]

get_transaction_receipt(*tx_hash: Union[Hash32, HexBytes, HexStr]*, *timeout=None*) → Optional[TxReceipt]

get_transaction_receipts(*tx_hashes: Sequence[Union[bytes, HexStr]]*) → List[Optional[TxReceipt]]

get_transactions(*tx_hashes: List[Union[Hash32, HexBytes, HexStr]]*) → List[Optional[TxData]]

is_contract(*contract_address: ChecksumAddress*) → bool

is_eip1559_supported() → EthereumNetwork

Returns

True if EIP1559 is supported by the node, *False* otherwise

property multicall: Multicall

static private_key_to_address(*private_key*)

send_eth_to(*private_key: str*, *to: str*, *gas_price: int*, *value: Wei*, *gas: Optional[int] = None*, *nonce: Optional[int] = None*, *retry: bool = False*, *block_identifier: Optional[Union[Literal['latest', 'earliest', 'pending'], BlockNumber, Hash32, HexStr, HexBytes, int]] = 'pending'*) → bytes

Send ether using configured account

Parameters

- **private_key** – to
- **to** – to
- **gas_price** – gas_price
- **value** – value(wei)
- **gas** – gas, defaults to 22000
- **retry** – Retry if a problem is found
- **nonce** – Nonce of sender account
- **block_identifier** – Block identifier for nonce calculation

Returns

tx_hash

send_raw_transaction(*raw_transaction: Union[bytes, HexStr]*) → HexBytes

send_transaction(*transaction_dict: TxParams*) → HexBytes

send_unsigned_transaction(*tx: TxParams*, *private_key: Optional[str] = None*, *public_key: Optional[str] = None*, *retry: bool = False*, *block_identifier: Optional[Union[Literal['latest', 'earliest', 'pending'], BlockNumber, Hash32, HexStr, HexBytes, int]] = 'pending'*) → HexBytes

Send a tx using an unlocked public key in the node or a private key. Both *public_key* and *private_key* cannot be *None*

Parameters

- **tx** –
- **private_key** –
- **public_key** –
- **retry** – Retry if a problem with nonce is found
- **block_identifier** – For nonce calculation, recommended is *pending*

Returns
tx hash

set_eip1559_fees(*tx: TxParams*, *tx_speed:* TxSpeed *= TxSpeed.NORMAL*) → TxParams

Returns
TxParams in EIP1559 format

Raises
ValueError if EIP1559 not supported

class gnosis.eth.ethereum_client.**EthereumClientManager**(*ethereum_client:* EthereumClient)

Bases: object

class gnosis.eth.ethereum_client.**EthereumClientProvider**

Bases: object

class gnosis.eth.ethereum_client.**EthereumTxSent**(*tx_hash*, *tx*, *contract_address*)

Bases: tuple

contract_address: Optional[str]

Alias for field number 2

tx: TxParams

Alias for field number 1

tx_hash: bytes

Alias for field number 0

class gnosis.eth.ethereum_client.**ParityManager**(*ethereum_client:* EthereumClient)

Bases: *EthereumClientManager*

filter_out_errored_traces(*internal_txs: Sequence[Dict[str, Any]]*) → Sequence[Dict[str, Any]]

Filter out errored transactions (traces that are errored or that have an errored parent)

Parameters
internal_txs – Traces for the SAME ethereum tx, sorted ascending by *trace_address sorted(t, key = lambda i: i['traceAddress'])*. It's the default output from methods returning *traces* like *trace_block* or *trace_transaction*

Returns
List of not errored traces

get_next_traces(*tx_hash: Union[Hash32, HexBytes, HexStr]*, *trace_address: Sequence[int]*, *remove_delegate_calls: bool = False*, *remove_calls: bool = False*) → List[Dict[str, Any]]

Parameters

- **tx_hash** –
- **trace_address** –
- **remove_delegate_calls** – If True remove delegate calls from result
- **remove_calls** – If True remove calls from result

Returns
Children for a trace, E.g. if address is [0, 1] and number_traces = 1, it will return [0, 1, x]

Raises
`ValueError` if tracing is not supported

get_previous_trace(*tx_hash: Union[Hash32, HexBytes, HexStr]*, *trace_address: Sequence[int]*, *number_traces: int = 1*, *skip_delegate_calls: bool = False*) → Optional[Dict[str, Any]]

Parameters

- **tx_hash** –
- **trace_address** –
- **number_traces** – Number of traces to skip, by default get the immediately previous one
- **skip_delegate_calls** – If True filter out delegate calls

Returns
Parent trace for a trace

Raises
`ValueError` if tracing is not supported

trace_block(*block_identifier: Union[Literal['latest', 'earliest', 'pending'], BlockNumber, Hash32, HexStr, HexBytes, int]*) → List[Dict[str, Any]]

trace_blocks(*block_identifiers: List[Union[Literal['latest', 'earliest', 'pending'], BlockNumber, Hash32, HexStr, HexBytes, int]]*) → List[List[Dict[str, Any]]]

trace_filter(*from_block: int = 1*, *to_block: Optional[int] = None*, *from_address: Optional[Sequence[ChecksumAddress]] = None*, *to_address: Optional[Sequence[ChecksumAddress]] = None*, *after: Optional[int] = None*, *count: Optional[int] = None*) → List[Dict[str, Any]]

Get events using `trace_filter` method

Parameters

- **from_block** – Quantity or Tag - (optional) From this block. *0* is not working, it needs to be >= *1*
- **to_block** – Quantity or Tag - (optional) To this block.
- **from_address** – Array - (optional) Sent from these addresses.
- **to_address** – Address - (optional) Sent to these addresses.
- **after** – Quantity - (optional) The offset trace number
- **count** – Quantity - (optional) Integer number of traces to display in a batch.

Returns


```
[
    {
        "action": {
            "callType": "call",
            "from": "0x32be343b94f860124dc4fee278fdcbd38c102d88",
            "gas": "0x4c40d",
            "input": "0x",
            "to": "0x8bbb73bcb5d553b5a556358d27625323fd781d37",
            "value": "0x3f0650ec47fd240000"
        },
        "blockHash":
↪"0x86df301bcdd8248d982dbf039f09faf792684e1aeee99d5b58b77d620008b80f",
        "blockNumber": 3068183,
        "result": {
            "gasUsed": "0x0",
            "output": "0x"
        },
        "subtraces": 0,
        "traceAddress": [],
        "transactionHash":
↪"0x3321a7708b1083130bd78da0d62ead9f6683033231617c9d268e2c7e3fa6c104",
        "transactionPosition": 3,
        "type": "call"
    },
    {
        "action": {
            "from": "0x3b169a0fb55ea0b6bafe54c272b1fe4983742bf7",
            "gas": "0x49b0b",
            "init":
↪"0x60806040523480156100105760008080fd5b506040516060a38038061060a8339810180604052810190808051906
↪",
            "value": "0x0"
        },
        "blockHash":
↪"0x03f9f64dfeb7807b5df608e6957dd4d521fd71685aac5533451d27f0abe03660",
        "blockNumber": 3793534,
        "result": {
            "address": "0x61a7cc907c47c133d5ff5b685407201951fcbd08",
            "code":
↪"0x6080604052600436106100
↪",
            "gasUsed": "0x4683f"
        },
        "subtraces": 2,
        "traceAddress": [],
        "transactionHash":
↪"0x6c7e8f8778d33d81b29c4bd7526ee50a4cea340d69eed6c89ada4e6fab731789",
        "transactionPosition": 1,
        "type": "create"
    },
    {
```


(continues on next page)

(continued from previous page)

```
        'action': {
            'address': '0x4440adafbc6c4e45c299451c0eedc7c8b98c14ac',
            'balance': '0x0',
            'refundAddress': '0x0000000000000000000000000000000000000000'
        },
        'blockHash':
↪'0x8512d367492371edf44ebcbbbd935bc434946dddc2b126cb558df5906012186c',
        'blockNumber': 7829689,
        'result': None,
        'subtraces': 0,
        'traceAddress': [0, 0, 0, 0, 0, 0],
        'transactionHash':
↪'0x5f7af6aa390f9f8dd79ee692c37cbde76bb7869768b1bac438b6d176c94f637d',
        'transactionPosition': 35,
        'type': 'suicide'
    }
]
```

trace_transaction(*tx_hash: Union[Hash32, HexBytes, HexStr]*) → List[Dict[str, Any]]

Parameters
tx_hash –

Returns
List of internal txs for *tx_hash*

trace_transactions(*tx_hashes: Sequence[Union[Hash32, HexBytes, HexStr]]*) → List[List[Dict[str, Any]]]

Parameters
tx_hashes –

Returns
For every *tx_hash* a list of internal txs (in the same order as the *tx_hashes* were provided)

class gnosis.eth.ethereum_client.**TokenBalance**(*token_address*, *balance*)

Bases: tuple

balance: int

Alias for field number 1

token_address: str

Alias for field number 0

class gnosis.eth.ethereum_client.**TxSpeed**(*value*)

Bases: Enum

An enumeration.

FAST = 4

FASTEST = 6

NORMAL = 3

SLOW = 2

SLOWEST = 0

VERY_FAST = 5

VERY_SLOW = 1

gnosis.eth.ethereum_client.**tx_with_exception_handling**(*func*)

Parity

- https://github.com/openethereum/openethereum/blob/main/rpc/src/v1/helpers/errors.rs

Geth

- https://github.com/ethereum/go-ethereum/blob/master/core/error.go
- https://github.com/ethereum/go-ethereum/blob/master/core/tx_pool.go

Comparison

- https://gist.github.com/kunal365roy/3c37ac9d1c3aaf31140f7c5faa083932

Parameters
func –

Returns

gnosis.eth.typing module

class gnosis.eth.typing.**BalanceDict**(**args*, ***kwargs*)

Bases: dict

balance: int

token_address: Optional[str]

gnosis.eth.utils module

gnosis.eth.utils.**compare_byte_code**(*code_1: bytes*, *code_2: bytes*) → bool

Compare code, removing swarm metadata if necessary

Parameters

- **code_1** –
- **code_2** –

Returns
True if same code, False otherwise

gnosis.eth.utils.**decode_string_or_bytes32**(*data: bytes*) → str

gnosis.eth.utils.**generate_address_2**(*from_: Union[str, bytes]*, *salt: Union[str, bytes]*, *init_code: Union[str, bytes]*) → str

Generates an address for a contract created using CREATE2.

Parameters

- **from** – The address which is creating this new address (need to be 20 bytes)
- **salt** – A salt (32 bytes)

- **init_code** – A init code of the contract being created

Returns
Address of the new contract

gnosis.eth.utils.**get_eth_address_with_invalid_checksum**() → str

gnosis.eth.utils.**get_eth_address_with_key**() → Tuple[str, bytes]

gnosis.eth.utils.**mk_contract_address**(*address: Union[str, bytes]*, *nonce: int*) → str

gnosis.eth.utils.**remove_swarm_metadata**(*code: bytes*) → bytes

Remove swarm metadata from Solidity bytecode

Parameters
code –

Returns
Code without metadata

Module contents

gnosis.safe package

Subpackages

Submodules

gnosis.safe.exceptions module

exception gnosis.safe.exceptions.**CannotEstimateGas**

Bases: *SafeServiceException*

exception gnosis.safe.exceptions.**CannotRetrieveSafeInfoException**

Bases: *SafeServiceException*

exception gnosis.safe.exceptions.**CouldNotFinishInitialization**

Bases: *InvalidMultisigTx*

exception gnosis.safe.exceptions.**CouldNotPayGasWithEther**

Bases: *InvalidMultisigTx*

exception gnosis.safe.exceptions.**CouldNotPayGasWithToken**

Bases: *InvalidMultisigTx*

exception gnosis.safe.exceptions.**HashHasNotBeenApproved**

Bases: *InvalidMultisigTx*

exception gnosis.safe.exceptions.**InvalidChecksumAddress**

Bases: *SafeServiceException*

exception gnosis.safe.exceptions.**InvalidContractSignatureLocation**

Bases: *InvalidMultisigTx*

exception gnosis.safe.exceptions.**InvalidInternalTx**

Bases: *InvalidMultisigTx*

exception gnosis.safe.exceptions.**InvalidMultisigTx**

Bases: *SafeServiceException*

exception gnosis.safe.exceptions.**InvalidOwnerProvided**

Bases: *InvalidMultisigTx*

exception gnosis.safe.exceptions.**InvalidPaymentToken**

Bases: *SafeServiceException*

exception gnosis.safe.exceptions.**InvalidSignaturesProvided**

Bases: *InvalidMultisigTx*

exception gnosis.safe.exceptions.**MethodCanOnlyBeCalledFromThisContract**

Bases: *InvalidMultisigTx*

exception gnosis.safe.exceptions.**ModuleManagerException**

Bases: *InvalidMultisigTx*

exception gnosis.safe.exceptions.**NotEnoughSafeTransactionGas**

Bases: *InvalidMultisigTx*

exception gnosis.safe.exceptions.**OnlyOwnersCanApproveAHash**

Bases: *InvalidMultisigTx*

exception gnosis.safe.exceptions.**OwnerManagerException**

Bases: *InvalidMultisigTx*

exception gnosis.safe.exceptions.**SafeServiceException**

Bases: Exception

exception gnosis.safe.exceptions.**SafeTransactionFailedWhenGasPriceAndSafeTxGasEmpty**

Bases: *InvalidMultisigTx*

exception gnosis.safe.exceptions.**SignatureNotProvidedByOwner**

Bases: *InvalidMultisigTx*

exception gnosis.safe.exceptions.**SignaturesDataTooShort**

Bases: *InvalidMultisigTx*

exception gnosis.safe.exceptions.**ThresholdNeedsToBeDefined**

Bases: *InvalidMultisigTx*

gnosis.safe.multi_send module

class gnosis.safe.multi_send.**MultiSend**(*address: str*, *ethereum_client:* EthereumClient)

Bases: object

build_tx_data(*multi_send_txs: List[*MultiSendTx*]*) → bytes

Txs don't need to be valid to get through

Parameters

- **multi_send_txs** –
- **sender** –

Returns

static deploy_contract(*ethereum_client: EthereumClient*, *deployer_account: LocalAccount*) → *EthereumTxSent*

Deploy proxy factory contract

Parameters

- **ethereum_client** –
- **deployer_account** – Ethereum Account

Returns
deployed contract address

dummy_w3 = <web3.main.Web3 object>

classmethod from_bytes(*encoded_multisend_txs: Union[str, bytes]*) → List[*MultiSendTx*]

Decodes one or more multisend transactions from *bytes transactions* (Abi decoded)

Parameters
encoded_multisend_txs –

Returns
List of MultiSendTxs

classmethod from_transaction_data(*multisend_data: Union[str, bytes]*) → List[*MultiSendTx*]

Decodes multisend transactions from transaction data (ABI encoded with selector)

Returns

get_contract()

class gnosis.safe.multi_send.**MultiSendOperation**(*value*)

Bases: Enum

An enumeration.

CALL = 0

DELEGATE_CALL = 1

class gnosis.safe.multi_send.**MultiSendTx**(*operation: MultiSendOperation*, *to: str*, *value: int*, *data: Union[bytes, HexStr]*, *old_encoding: bool = False*)

Bases: object

Wrapper for a single MultiSendTx

property data_length: int

property encoded_data

classmethod from_bytes(*encoded_multisend_tx: Union[str, bytes]*) → *MultiSendTx*

Decoded one MultiSend transaction. ABI must be used to get the *transactions* parameter and use that data for this function :param encoded_multisend_tx: :return:

gnosis.safe.proxy_factory module

class gnosis.safe.proxy_factory.**ProxyFactory**(*address: ChecksumAddress*, *ethereum_client:* EthereumClient)

Bases: object

check_proxy_code(*address: ChecksumAddress*) → bool

Check if proxy is valid :param address: Ethereum address to check :return: True if proxy is valid, False otherwise

deploy_proxy_contract(*deployer_account: LocalAccount*, *master_copy: ChecksumAddress*, *initializer: bytes = b''*, *gas: Optional[int] = None*, *gas_price: Optional[int] = None*) → *EthereumTxSent*

Deploy proxy contract via ProxyFactory using *createProxy* function :param deployer_account: Ethereum account :param master_copy: Address the proxy will point at :param initializer: Initializer :param gas: Gas :param gas_price: Gas Price :return: EthereumTxSent

deploy_proxy_contract_with_nonce(*deployer_account: LocalAccount*, *master_copy: ChecksumAddress*, *initializer: bytes*, *salt_nonce: int*, *gas: Optional[int] = None*, *gas_price: Optional[int] = None*, *nonce: Optional[int] = None*) → *EthereumTxSent*

Deploy proxy contract via Proxy Factory using *createProxyWithNonce* (create2)

Parameters

- **deployer_account** – Ethereum account
- **master_copy** – Address the proxy will point at
- **initializer** – Data for safe creation
- **salt_nonce** – Uint256 for *create2* salt
- **gas** – Gas
- **gas_price** – Gas Price
- **nonce** – Nonce

Returns
Tuple(tx-hash, tx, deployed contract address)

classmethod **deploy_proxy_factory_contract**(*ethereum_client:* EthereumClient, *deployer_account: LocalAccount*) → *EthereumTxSent*

Deploy proxy factory contract last version (v1.3.0)

Parameters

- **ethereum_client** –
- **deployer_account** – Ethereum Account

Returns
deployed contract address

classmethod **deploy_proxy_factory_contract_v1_0_0**(*ethereum_client:* EthereumClient, *deployer_account: LocalAccount*) → *EthereumTxSent*

Deploy proxy factory contract v1.0.0

Parameters

- **ethereum_client** –
- **deployer_account** – Ethereum Account

Returns
deployed contract address

classmethod deploy_proxy_factory_contract_v1_1_1(*ethereum_client: EthereumClient*, *deployer_account: LocalAccount*) → *EthereumTxSent*

Deploy proxy factory contract v1.1.1

Parameters

- **ethereum_client** –
- **deployer_account** – Ethereum Account

Returns
deployed contract address

get_contract(*address: Optional[ChecksumAddress] = None*)

get_proxy_runtime_code(*address: Optional[ChecksumAddress] = None*)

Get runtime code for current proxy factory

gnosis.safe.safe module

class gnosis.safe.safe.**Safe**(*address: ChecksumAddress*, *ethereum_client: EthereumClient*)

Bases: object

Class to manage a Gnosis Safe

FALLBACK_HANDLER_STORAGE_SLOT = 49122629484629529244014240937346711770925847994644146912111677022347558721749

GUARD_STORAGE_SLOT = 33528237782592280163068556224972516439282563014722366175641814928123294921928

build_multisig_tx(*to: str*, *value: int*, *data: bytes*, *operation: int = 0*, *safe_tx_gas: int = 0*, *base_gas: int = 0*, *gas_price: int = 0*, *gas_token: str = '0x0000000000000000000000000000000000000000'*, *refund_receiver: str = '0x0000000000000000000000000000000000000000'*, *signatures: bytes = b''*, *safe_nonce: Optional[int] = None*, *safe_version: Optional[str] = None*) → *SafeTx*

Allows to execute a Safe transaction confirmed by required number of owners and then pays the account that submitted the transaction. The fees are always transfered, even if the user transaction fails

Parameters

- **to** – Destination address of Safe transaction
- **value** – Ether value of Safe transaction
- **data** – Data payload of Safe transaction
- **operation** – Operation type of Safe transaction
- **safe_tx_gas** – Gas that should be used for the Safe transaction
- **base_gas** – Gas costs for that are independent of the transaction execution (e.g. base transaction fee, signature check, payment of the refund)

- **gas_price** – Gas price that should be used for the payment calculation
- **gas_token** – Token address (or *0x000..000* if ETH) that is used for the payment
- **refund_receiver** – Address of receiver of gas payment (or *0x000..000* if tx.origin).
- **signatures** – Packed signature data ({bytes32 r}{bytes32 s}{uint8 v})
- **safe_nonce** – Nonce of the safe (to calculate hash)
- **safe_version** – Safe version (to calculate hash)

Returns

static build_safe_create2_tx(*ethereum_client: EthereumClient*, *master_copy_address: str*, *proxy_factory_address: str*, *salt_nonce: int*, *owners: List[str]*, *threshold: int*, *gas_price: int*, *payment_token: Optional[str]*, *payment_receiver: Optional[str] = None*, *fallback_handler: Optional[str] = '0x0000000000000000000000000000000000000000'*, *payment_token_eth_value: float = 1.0*, *fixed_creation_cost: Optional[int] = None*) → *SafeCreate2Tx*

Prepare safe proxy deployment for being relayed. It calculates and sets the costs of deployment to be returned to the sender of the tx. If you are an advanced user you may prefer to use *create* function

static build_safe_creation_tx(*ethereum_client: EthereumClient*, *master_copy_old_address: str*, *s: int*, *owners: List[str]*, *threshold: int*, *gas_price: int*, *payment_token: Optional[str]*, *payment_receiver: str*, *payment_token_eth_value: float = 1.0*, *fixed_creation_cost: Optional[int] = None*) → *SafeCreationTx*

check_funds_for_tx_gas(*safe_tx_gas: int*, *base_gas: int*, *gas_price: int*, *gas_token: str*) → bool

Check safe has enough funds to pay for a tx

Parameters

- **safe_tx_gas** – Safe tx gas
- **base_gas** – Data gas
- **gas_price** – Gas Price
- **gas_token** – Gas Token, to use token instead of ether for the gas

Returns

True if enough funds, *False* otherwise

static create(*ethereum_client: EthereumClient*, *deployer_account: LocalAccount*, *master_copy_address: str*, *owners: List[str]*, *threshold: int*, *fallback_handler: str = '0x0000000000000000000000000000000000000000'*, *proxy_factory_address: Optional[str] = None*, *payment_token: str = '0x0000000000000000000000000000000000000000'*, *payment: int = 0*, *payment_receiver: str = '0x0000000000000000000000000000000000000000'*) → *EthereumTxSent*

Deploy new Safe proxy pointing to the specified *master_copy* address and configured with the provided *owners* and *threshold*. By default, payment for the deployer of the tx will be *0*. If *proxy_factory_address* is set deployment will be done using the proxy factory instead of calling the *constructor* of a new *DelegatedProxy* Using *proxy_factory_address* is recommended, as it takes less gas. (Testing with *Ganache* and 1 owner 261534 without proxy vs 229022 with Proxy)

classmethod deploy_compatibility_fallback_handler(*ethereum_client: EthereumClient*, *deployer_account: LocalAccount*) → *EthereumTxSent*

Deploy Compatibility Fallback handler v1.3.0

Parameters

- **ethereum_client** –
- **deployer_account** – Ethereum account

Returns
deployed contract address

static deploy_master_contract_v0_0_1(*ethereum_client:* EthereumClient, *deployer_account:* *LocalAccount*) → *EthereumTxSent*

Deploy master contract. Takes deployer_account (if unlocked in the node) or the deployer private key

Parameters

- **ethereum_client** –
- **deployer_account** – Ethereum account

Returns
deployed contract address

static deploy_master_contract_v1_0_0(*ethereum_client:* EthereumClient, *deployer_account:* *LocalAccount*) → *EthereumTxSent*

Deploy master contract. Takes deployer_account (if unlocked in the node) or the deployer private key

Parameters

- **ethereum_client** –
- **deployer_account** – Ethereum account

Returns
deployed contract address

classmethod deploy_master_contract_v1_1_1(*ethereum_client:* EthereumClient, *deployer_account:* *LocalAccount*) → *EthereumTxSent*

Deploy master contract v1.1.1. Takes deployer_account (if unlocked in the node) or the deployer private key Safe with version > v1.1.1 doesn't need to be initialized as it already has a constructor

Parameters

- **ethereum_client** –
- **deployer_account** – Ethereum account

Returns
deployed contract address

classmethod deploy_master_contract_v1_3_0(*ethereum_client:* EthereumClient, *deployer_account:* *LocalAccount*) → *EthereumTxSent*

Deploy master contract v1.3.0. Takes deployer_account (if unlocked in the node) or the deployer private key Safe with version > v1.1.1 doesn't need to be initialized as it already has a constructor

Parameters

- **ethereum_client** –
- **deployer_account** – Ethereum account

Returns
deployed contract address

static estimate_safe_creation(*ethereum_client: EthereumClient*, *old_master_copy_address: str*, *number_owners: int*, *gas_price: int*, *payment_token: Optional[str]*, *payment_receiver: str = '0x0000000000000000000000000000000000000000'*, *payment_token_eth_value: float = 1.0*, *fixed_creation_cost: Optional[int] = None*) → *SafeCreationEstimate*

static estimate_safe_creation_2(*ethereum_client: EthereumClient*, *master_copy_address: str*, *proxy_factory_address: str*, *number_owners: int*, *gas_price: int*, *payment_token: Optional[str]*, *payment_receiver: str = '0x0000000000000000000000000000000000000000'*, *fallback_handler: Optional[str] = None*, *payment_token_eth_value: float = 1.0*, *fixed_creation_cost: Optional[int] = None*) → *SafeCreationEstimate*

estimate_tx_base_gas(*to: str*, *value: int*, *data: bytes*, *operation: int*, *gas_token: str*, *estimated_tx_gas: int*) → int

Calculate gas costs that are independent of the transaction execution(e.g. base transaction fee, signature check, payment of the refund...)

Parameters

- **to** –
- **value** –
- **data** –
- **operation** –
- **gas_token** –
- **estimated_tx_gas** – gas calculated with *estimate_tx_gas*

Returns

estimate_tx_gas(*to: str*, *value: int*, *data: bytes*, *operation: int*) → int

Estimate tx gas. Use *requiredTxGas* on the Safe contract and fallbacks to *eth_estimateGas* if that method fails. Note: *eth_estimateGas* cannot estimate delegate calls

Parameters

- **to** –
- **value** –
- **data** –
- **operation** –

Returns
Estimated gas for Safe inner tx

Raises
CannotEstimateGas

estimate_tx_gas_by_trying(*to: str*, *value: int*, *data: Union[bytes, str]*, *operation: int*)

Try to get an estimation with Safe's *requiredTxGas*. If estimation if successful, try to set a gas limit and estimate again. If gas estimation is ok, same gas estimation should be returned, if it's less than required estimation will not be completed, so estimation was not accurate and gas limit needs to be increased.

Parameters

- **to** –
- **value** –
- **data** –
- **operation** –

Returns
Estimated gas calling *requiredTxGas* setting a gas limit and checking if *eth_call* is successful

Raises
CannotEstimateGas

estimate_tx_gas_with_safe(*to: str*, *value: int*, *data: bytes*, *operation: int*, *gas_limit: Optional[int] = None*, *block_identifier: Optional[Union[Literal['latest', 'earliest', 'pending'], BlockNumber, Hash32, HexStr, HexBytes, int]] = 'latest'*) → int

Estimate tx gas using safe *requiredTxGas* method

Returns
int: Estimated gas

Raises
CannotEstimateGas: If gas cannot be estimated

Raises
ValueError: Cannot decode received data

estimate_tx_gas_with_web3(*to: str*, *value: int*, *data: Union[bytes, HexStr]*) → int

Parameters

- **to** –
- **value** –
- **data** –

Returns
Estimation using web3 *estimate_gas*

estimate_tx_operational_gas(*data_bytes_length: int*)

DEPRECATED. *estimate_tx_base_gas* already includes this. Estimates the gas for the verification of the signatures and other safe related tasks before and after executing a transaction. Calculation will be the sum of:

- Base cost of 15000 gas
- 100 of gas per word of *data_bytes*
- Validate the signatures 5000 * threshold (ecrecover for ecdsa ~= 4K gas)

Parameters
data_bytes_length – Length of the data (in bytes, so *len(HexBytes('0x12'))* would be *1*

Returns
gas costs per signature * threshold of Safe

get_contract() → Contract

retrieve_all_info(*block_identifier: Optional[Union[Literal['latest', 'earliest', 'pending'], BlockNumber, Hash32, HexStr, HexBytes, int]] = 'latest'*) → *SafeInfo*

Get all Safe info in the same batch call.

Parameters
block_identifier –

Returns

Raises
CannotRetrieveSafeInfoException

retrieve_code() → HexBytes

retrieve_fallback_handler(*block_identifier: Optional[Union[Literal['latest', 'earliest', 'pending'], BlockNumber, Hash32, HexStr, HexBytes, int]] = 'latest'*) → ChecksumAddress

retrieve_guard(*block_identifier: Optional[Union[Literal['latest', 'earliest', 'pending'], BlockNumber, Hash32, HexStr, HexBytes, int]] = 'latest'*) → ChecksumAddress

retrieve_is_hash_approved(*owner: str*, *safe_hash: bytes*, *block_identifier: Optional[Union[Literal['latest', 'earliest', 'pending'], BlockNumber, Hash32, HexStr, HexBytes, int]] = 'latest'*) → bool

retrieve_is_message_signed(*message_hash: bytes*, *block_identifier: Optional[Union[Literal['latest', 'earliest', 'pending'], BlockNumber, Hash32, HexStr, HexBytes, int]] = 'latest'*) → bool

retrieve_is_owner(*owner: str*, *block_identifier: Optional[Union[Literal['latest', 'earliest', 'pending'], BlockNumber, Hash32, HexStr, HexBytes, int]] = 'latest'*) → bool

retrieve_master_copy_address(*block_identifier: Optional[Union[Literal['latest', 'earliest', 'pending'], BlockNumber, Hash32, HexStr, HexBytes, int]] = 'latest'*) → ChecksumAddress

retrieve_modules(*pagination: Optional[int] = 50*, *block_identifier: Optional[Union[Literal['latest', 'earliest', 'pending'], BlockNumber, Hash32, HexStr, HexBytes, int]] = 'latest'*) → List[str]

Parameters

- **pagination** – Number of modules to get per request
- **block_identifier** –

Returns
List of module addresses

retrieve_nonce(*block_identifier: Optional[Union[Literal['latest', 'earliest', 'pending'], BlockNumber, Hash32, HexStr, HexBytes, int]] = 'latest'*) → int

retrieve_owners(*block_identifier: Optional[Union[Literal['latest', 'earliest', 'pending'], BlockNumber, Hash32, HexStr, HexBytes, int]] = 'latest'*) → List[str]

retrieve_threshold(*block_identifier: Optional[Union[Literal['latest', 'earliest', 'pending'], BlockNumber, Hash32, HexStr, HexBytes, int]] = 'latest'*) → int

retrieve_version(*block_identifier: Optional[Union[Literal['latest', 'earliest', 'pending'], BlockNumber, Hash32, HexStr, HexBytes, int]] = 'latest'*) → str

send_multisig_tx(*to: str*, *value: int*, *data: bytes*, *operation: int*, *safe_tx_gas: int*, *base_gas: int*, *gas_price: int*, *gas_token: str*, *refund_receiver: str*, *signatures: bytes*, *tx_sender_private_key: str*, *tx_gas=None*, *tx_gas_price=None*, *block_identifier: Optional[Union[Literal['latest', 'earliest', 'pending'], BlockNumber, Hash32, HexStr, HexBytes, int]] = 'latest'*) → *EthereumTxSent*

Build and send Safe tx

Parameters

- **to** –
- **value** –
- **data** –
- **operation** –
- **safe_tx_gas** –
- **base_gas** –
- **gas_price** –
- **gas_token** –
- **refund_receiver** –
- **signatures** –
- **tx_sender_private_key** –
- **tx_gas** – Gas for the external tx. If not, *(safe_tx_gas + data_gas) * 2* will be used
- **tx_gas_price** – Gas price of the external tx. If not, *gas_price* will be used
- **block_identifier** –

Returns

Tuple(tx_hash, tx)

Raises

InvalidMultisigTx: If user tx cannot go through the Safe

class gnosis.safe.safe.**SafeCreationEstimate**(*gas*, *gas_price*, *payment*, *payment_token*)

Bases: tuple

gas: int

Alias for field number 0

gas_price: int

Alias for field number 1

payment: int

Alias for field number 2

payment_token: Optional[str]

Alias for field number 3

class gnosis.safe.safe.**SafeInfo**(*address: <function NewType.<locals>.new_type at 0x7f907d8bb4c0>, fallback_handler: <function NewType.<locals>.new_type at 0x7f907d8bb4c0>, guard: <function NewType.<locals>.new_type at 0x7f907d8bb4c0>, master_copy: <function NewType.<locals>.new_type at 0x7f907d8bb4c0>, modules: List[ChecksumAddress], nonce: int, owners: List[ChecksumAddress], threshold: int, version: str*)

Bases: object

address: ChecksumAddress

fallback_handler: ChecksumAddress

guard: ChecksumAddress

master_copy: ChecksumAddress

modules: List[ChecksumAddress]

nonce: int

owners: List[ChecksumAddress]

threshold: int

version: str

class gnosis.safe.safe.**SafeOperation**(*value*)

Bases: Enum

An enumeration.

CALL = 0

CREATE = 2

DELEGATE_CALL = 1

gnosis.safe.safe_create2_tx module

exception gnosis.safe.safe_create2_tx.**InvalidERC20Token**

Bases: Exception

class gnosis.safe.safe_create2_tx.**SafeCreate2Tx**(*salt_nonce*, *owners*, *threshold*, *fallback_handler*, *master_copy_address*, *proxy_factory_address*, *payment_receiver*, *payment_token*, *payment*, *gas*, *gas_price*, *payment_token_eth_value*, *fixed_creation_cost*, *safe_address*, *safe_setup_data*)

Bases: tuple

fallback_handler: str

Alias for field number 3

fixed_creation_cost: Optional[int]

Alias for field number 12

gas: int

Alias for field number 9

gas_price: int

Alias for field number 10

master_copy_address: str

Alias for field number 4

owners: List[str]

Alias for field number 1

payment: int

Alias for field number 8

property payment_ether

payment_receiver: str

Alias for field number 6

payment_token: str

Alias for field number 7

payment_token_eth_value: float

Alias for field number 11

proxy_factory_address: str

Alias for field number 5

safe_address: str

Alias for field number 13

safe_setup_data: bytes

Alias for field number 14

salt_nonce: int

Alias for field number 0

threshold: int

Alias for field number 2

class gnosis.safe.safe_create2_tx.**SafeCreate2TxBuilder**(*w3: Web3*, *master_copy_address: str*, *proxy_factory_address: str*)

Bases: object

build(*owners: List[str]*, *threshold: int*, *salt_nonce: int*, *gas_price: int*, *fallback_handler: Optional[str] = None*, *payment_receiver: Optional[str] = None*, *payment_token: Optional[str] = None*, *payment_token_eth_value: float = 1.0*, *fixed_creation_cost: Optional[int] = None*)

Prepare Safe creation :param owners: Owners of the Safe :param threshold: Minimum number of users required to operate the Safe :param fallback_handler: Handler for fallback calls to the Safe :param salt_nonce: Web3 instance :param gas_price: Gas Price :param payment_receiver: Address to refund when the Safe is created. Address(0) if no need to refund :param payment_token: Payment token instead of paying the funder with ether. If None Ether will be used :param payment_token_eth_value: Value of payment token per 1 Ether :param fixed_creation_cost: Fixed creation cost of Safe (Wei)

calculate_create2_address(*safe_setup_data: bytes*, *salt_nonce: int*)

gnosis.safe.safe_creation_tx module

exception gnosis.safe.safe_creation_tx.**InvalidERC20Token**

Bases: Exception

class gnosis.safe.safe_creation_tx.**SafeCreationTx**(*w3: Web3*, *owners: List[str]*, *threshold: int*, *signature_s: int*, *master_copy: str*, *gas_price: int*, *funder: Optional[str]*, *payment_token: Optional[str] = None*, *payment_token_eth_value: float = 1.0*, *fixed_creation_cost: Optional[int] = None*)

Bases: object

static find_valid_random_signature(*s: int*) → Tuple[int, int]

Find v and r valid values for a given s :param s: random value :return: v, r

property payment_ether

gnosis.safe.safe_signature module

exception gnosis.safe.safe_signature.**CannotCheckEIP1271ContractSignature**

Bases: *SafeSignatureException*

class gnosis.safe.safe_signature.**SafeSignature**(*signature: Union[bytes, str]*, *safe_tx_hash: Union[bytes, str]*)

Bases: ABC

export_signature() → HexBytes

Exports signature in a format that's valid individually. That's important for contract signatures, as it will fix the offset :return:

abstract is_valid(*ethereum_client:* EthereumClient, *safe_address: str*) → bool

Parameters

- **ethereum_client** – Required for Contract Signature and Approved Hash check
- **safe_address** – Required for Approved Hash check

Returns

True if signature is valid, *False* otherwise

abstract property owner

Returns

Decode owner from signature, without any further validation (signature can be not valid)

classmethod parse_signature(*signatures: Union[bytes, str]*, *safe_tx_hash: Union[bytes, str]*, *ignore_trailing: bool = True*) → List[*SafeSignature*]

Parameters

- **signatures** – One or more signatures appended. EIP1271 data at the end is supported.
- **safe_tx_hash** –
- **ignore_trailing** – Ignore trailing data on the signature. Some libraries pad it and add some zeroes at the end

Returns
List of SafeSignatures decoded

abstract property signature_type: *SafeSignatureType*

class gnosis.safe.safe_signature.**SafeSignatureApprovedHash**(*signature: Union[bytes, str]*, *safe_tx_hash: Union[bytes, str]*)

Bases: *SafeSignature*

classmethod build_for_owner(*owner: str*, *safe_tx_hash: str*) → *SafeSignatureApprovedHash*

is_valid(*ethereum_client:* EthereumClient, *safe_address: str*) → bool

Parameters

- **ethereum_client** – Required for Contract Signature and Approved Hash check
- **safe_address** – Required for Approved Hash check

Returns
True if signature is valid, *False* otherwise

property owner

Returns
Decode owner from signature, without any further validation (signature can be not valid)

property signature_type

class gnosis.safe.safe_signature.**SafeSignatureContract**(*signature: Union[bytes, str]*, *safe_tx_hash: Union[bytes, str]*, *contract_signature: Union[bytes, str]*)

Bases: *SafeSignature*

EIP1271_MAGIC_VALUE = HexBytes('0x20c13b0b')

EIP1271_MAGIC_VALUE_UPDATED = HexBytes('0x1626ba7e')

export_signature() → HexBytes

Fix offset (s) and append *contract_signature* at the end of the signature :return:

is_valid(*ethereum_client:* EthereumClient, **args*) → bool

Parameters

- **ethereum_client** – Required for Contract Signature and Approved Hash check
- **safe_address** – Required for Approved Hash check

Returns
True if signature is valid, *False* otherwise

property owner: ChecksumAddress

Returns
Address of contract signing. No further checks to get the owner are needed, but it could be a non existing contract

property signature_type: *SafeSignatureType*

class gnosis.safe.safe_signature.**SafeSignatureEOA**(*signature: Union[bytes, str]*, *safe_tx_hash: Union[bytes, str]*)

Bases: *SafeSignature*

is_valid(**args*) → bool

Parameters

- **ethereum_client** – Required for Contract Signature and Approved Hash check
- **safe_address** – Required for Approved Hash check

Returns
True if signature is valid, *False* otherwise

property owner

Returns
Decode owner from signature, without any further validation (signature can be not valid)

property signature_type

class gnosis.safe.safe_signature.**SafeSignatureEthSign**(*signature: Union[bytes, str]*, *safe_tx_hash: Union[bytes, str]*)

Bases: *SafeSignature*

is_valid(**args*) → bool

Parameters

- **ethereum_client** – Required for Contract Signature and Approved Hash check
- **safe_address** – Required for Approved Hash check

Returns
True if signature is valid, *False* otherwise

property owner

Returns
Decode owner from signature, without any further validation (signature can be not valid)

property signature_type

exception gnosis.safe.safe_signature.**SafeSignatureException**

Bases: Exception

class gnosis.safe.safe_signature.**SafeSignatureType**(*value*)

Bases: Enum

An enumeration.

APPROVED_HASH = 1

CONTRACT_SIGNATURE = 0

EOA = 2

ETH_SIGN = 3

static from_v(*v: int*)

gnosis.safe.safe_signature.**uint_to_address**(*value: int*) → ChecksumAddress

Convert a Solidity *uint* value to a checksummed *address*, removing invalid padding bytes if present

Returns

Checksummed address

gnosis.safe.safe_tx module

class gnosis.safe.safe_tx.**EIP712LegacySafeTx**(***kwargs*)

Bases: EIP712Struct

data = <eip712_structs.types.Bytes object>

dataGas = <eip712_structs.types.Uint object>

gasPrice = <eip712_structs.types.Uint object>

gasToken = <eip712_structs.types.Address object>

nonce = <eip712_structs.types.Uint object>

operation = <eip712_structs.types.Uint object>

refundReceiver = <eip712_structs.types.Address object>

safeTxGas = <eip712_structs.types.Uint object>

to = <eip712_structs.types.Address object>

type_name = 'SafeTx'

value = <eip712_structs.types.Uint object>

class gnosis.safe.safe_tx.**EIP712SafeTx**(***kwargs*)

Bases: EIP712Struct

baseGas = <eip712_structs.types.Uint object>

data = <eip712_structs.types.Bytes object>

gasPrice = <eip712_structs.types.Uint object>

gasToken = <eip712_structs.types.Address object>

nonce = <eip712_structs.types.Uint object>

operation = <eip712_structs.types.Uint object>

refundReceiver = <eip712_structs.types.Address object>

safeTxGas = <eip712_structs.types.Uint object>

to = <eip712_structs.types.Address object>

type_name = 'SafeTx'

value = <eip712_structs.types.Uint object>

class gnosis.safe.safe_tx.**SafeTx**(*ethereum_client: EthereumClient*, *safe_address: str*, *to: Optional[str]*, *value: int*, *data: bytes*, *operation: int*, *safe_tx_gas: int*, *base_gas: int*, *gas_price: int*, *gas_token: Optional[str]*, *refund_receiver: Optional[str]*, *signatures: Optional[bytes] = None*, *safe_nonce: Optional[int] = None*, *safe_version: Optional[str] = None*, *chain_id: Optional[int] = None*)

Bases: object

call(*tx_sender_address: Optional[str] = None*, *tx_gas: Optional[int] = None*, *block_identifier: Optional[Union[Literal['latest', 'earliest', 'pending'], BlockNumber, Hash32, HexStr, HexBytes, int]] = 'latest'*) → int

Parameters

- **tx_sender_address** –
- **tx_gas** – Force a gas limit
- **block_identifier** –

Returns
1 if everything ok

property chain_id: int

property contract

property eip712_structured_data: Dict

execute(*tx_sender_private_key: str*, *tx_gas: Optional[int] = None*, *tx_gas_price: Optional[int] = None*, *tx_nonce: Optional[int] = None*, *block_identifier: Optional[Union[Literal['latest', 'earliest', 'pending'], BlockNumber, Hash32, HexStr, HexBytes, int]] = 'latest'*, *eip1559_speed: Optional[TxSpeed] = None*) → Tuple[HexBytes, TxParams]

Send multisig tx to the Safe

Parameters

- **tx_sender_private_key** – Sender private key
- **tx_gas** – Gas for the external tx. If not, *(safe_tx_gas + base_gas) * 2* will be used
- **tx_gas_price** – Gas price of the external tx. If not, *gas_price* will be used
- **tx_nonce** – Force nonce for *tx_sender*
- **block_identifier** – *latest* or *pending*
- **eip1559_speed** – If provided, use EIP1559 transaction

Returns
Tuple(tx_hash, tx)

Raises
InvalidMultisigTx: If user tx cannot go through the Safe

recommended_gas() → Wei

Returns
Recommended gas to use on the ethereum_tx

property safe_nonce: str

property safe_tx_hash: HexBytes

property safe_version: str

sign(*private_key: str*) → bytes

{bytes32 r}{bytes32 s}{uint8 v} :param private_key: :return: Signature

property signers: List[str]

property sorted_signers

tx: TxParams

tx_hash: bytes

unsign(*address: str*) → bool

property w3

property w3_tx

Returns

Web3 contract tx prepared for *call*, *transact* or *buildTransaction*

gnosis.safe.serializers module

class gnosis.safe.serializers.**SafeMultisigEstimateTxSerializer**(**args*, ***kwargs*)

Bases: Serializer

validate(*data*)

validate_operation(*value*)

class gnosis.safe.serializers.**SafeMultisigTxSerializer**(**args*, ***kwargs*)

Bases: *SafeMultisigEstimateTxSerializer*

DEPRECATED, use *SafeMultisigTxSerializerV1* instead

class gnosis.safe.serializers.**SafeMultisigTxSerializerV1**(**args*, ***kwargs*)

Bases: *SafeMultisigEstimateTxSerializer*

Version 1.0.0 of the Safe changes *data_gas* to *base_gas*

class gnosis.safe.serializers.**SafeSignatureSerializer**(**args*, ***kwargs*)

Bases: Serializer

When using safe signatures *v* can have more values

check_r(*r*)

check_s(*s*)

check_v(*v*)

validate(*data*)

validate_v(*v*)

gnosis.safe.signatures module

gnosis.safe.signatures.**get_signing_address**(*signed_hash: Union[bytes, str]*, *v: int*, *r: int*, *s: int*) → str

Returns
checksummed ethereum address, for example *0x568c93675A8dEb121700A6FAdDdfE7DFAb66Ae4A*

Return type
str or *NULL_ADDRESS* if signature is not valid

gnosis.safe.signatures.**signature_split**(*signatures: Union[bytes, str]*, *pos: int = 0*) → Tuple[int, int, int]

Parameters

- **signatures** – signatures in form of {bytes32 r}{bytes32 s}{uint8 v}
- **pos** – position of the signature

Returns
Tuple with v, r, s

gnosis.safe.signatures.**signature_to_bytes**(*v: int*, *r: int*, *s: int*) → bytes

Convert ecdsa signature to bytes :param v: :param r: :param s: :return: signature in form of {bytes32 r}{bytes32 s}{uint8 v}

gnosis.safe.signatures.**signatures_to_bytes**(*signatures: List[Tuple[int, int, int]]*) → bytes

Convert signatures to bytes :param signatures: list of tuples(v, r, s) :return: 65 bytes per signature

Module contents

Module contents

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

g

INDEX

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y

Z