KrakenExApi

Release 0.0.1

Erik Körner

Jan 06, 2021

CONTENTS

CHAPTER

ONE

OVERVIEW

docs	
tests	

A Kraken Exchange API adapter.

- Free software: MIT license

1.1 Installation

```
pip install krakenexapi
```

You can also install the in-development version with:

```
pip install https://github.com/Querela/python-krakenexapi/archive/master.zip
```

1.2 Documentation

https://python-krakenexapi.readthedocs.io/

1.3 Development

To run all the tests run:

```
tox
```

Note, to combine the coverage data from all the tox environments run:

Windows	`set PYTEST_ADDOPTS=--cov-append` `tox`
Other	`PYTEST_ADDOPTS=--cov-append tox`

CHAPTER

TWO

INSTALLATION

At the command line:

```
pip install krakenexapi
```

CHAPTER

THREE

USAGE

To use KrakenExApi in a project:

```
import krakenexapi.api as kea
import krakenexapi.wallet as kew

api = kea.BasicKrakenExAPI(tier="Intermediate")
api.load_key()
kew.CurrencyPair.build_from_api(api)

wallet = kew.Wallet(api)
wallet._update()
```

CHAPTER

FOUR

REFERENCE

4.1 krakenexapi

4.2 krakenexapi.api

4.2.1 Raw API

class krakenexapi.api.**RawKrakenExAPI**(*key: Optional[str] = None*, *secret: Optional[str] = None*)

Bases: object

Raw Kraken Exchange API adspter.

Variables **session** (*requests.Session*) – requests session object (stores User-Agent)

api_domain = 'https://api.kraken.com'

load_key(*path: Optional[os.PathLike] = None*)

Load private Kraken Exchange API key/secret.

Search order:

1. *path* if file, then load directly,
2. *path* is folder, append kraken.key and load,
3. no *path*, try to load locally from kraken.key.

Raise NoPrivateKey, if

- no key file could be found,
- not both key/secret are found,
- secret is no valid base64.

Key file format:

```
key=your-key
secret=your-secret
```

Whitespaces will be stripped from front/end and around the equal sign.

Parameters **path** (*Optional[PathLike], optional*) – Path to kraken.key file, by default None

Raises ***NoPrivateKey*** – If no key file could be found or key not valid (format).

nonce() → int

Nonce for API request. Should be monotonic.

Returns timestamps seconds + milliseconds. Substracts seconds since 2021 (makes the nonce smaller).

Returns *int*

_query_raw(*path: str*, *data: Dict[str, Any]*, *headers: Dict[str, Any]*, *timeout: Optional[Tuple[int, float]] = None*) → Dict[str, Any]

_sign (*api_path: str*, *data: Dict[str, Any]*) → str
Create signature for private Kraken Exchange API request.

Parameters
- **api_path** (*str*) – API path (prefix + method)
- **data** (*Dict[str, Any]*) – API request data, must contain `nonce`

Returns *str* – signature

Notes

See Kraken Exchange API Docs, Example algorithm, Example client.

query_public (*method: str*, ***kwargs*) → Dict[str, Any]

query_private (*method: str*, *otp: Optional[str] = None*, ***kwargs*) → Dict[str, Any]

krakenexapi.api.**NONCE_OFFSET = -1609459200.0**
Nonce value *offset*, nonce value will start from year `2021`

krakenexapi.api.**API_METHODS_PUBLIC = ['Time', 'SystemStatus', 'Assets', 'AssetPairs', 'Tic**
List of allowed public endpoints

krakenexapi.api.**API_METHODS_PRIVATE = ['Balance', 'TradeBalance', 'TradeVolume', 'DepositMe**
List of allowed private endpoints

krakenexapi.api.**API_METHODS_NO_RETRY = ['AddOrder', 'AddExport', 'Withdraw', 'WalletTransfe**
List of API methods where we do not want to retry. e. g. no repeated AddOrder because it might create duplicate orders.

Notes

The *nonce()* will use an offset of *NONCE_OFFSET* for its value - so, the nonce value is comparatively smaller compared to the standard unix timestamp. Try to avoid using the same API key for different applications!

4.2.2 Basic Kraken Exchange API methods

Wraps the endpoints in *API_METHODS_PUBLIC* and *API_METHODS_PRIVATE* with simplified parameters and corrected return values. It will try to call rate limit both public and, when provided a `tier` (verification level), private API endpoints to avoid possible blacklisting. The full description of methods, parameters and return values csn be found in the Official Kraken REST API.

class krakenexapi.api.**BasicKrakenExAPI**(*key: Optional[str] = None*, *secret: Optional[str] = None*, *tier: Optional[str] = 'Starter'*)
Bases: *krakenexapi.api.BasicKrakenExAPIPublicMethods*, *krakenexapi.api.BasicKrakenExAPIPrivateUserDataMethods*, *krakenexapi.api.BasicKrakenExAPIPrivateUserTradingMethods*, *krakenexapi.api.BasicKrakenExAPIPrivateUserFundingMethods*, *krakenexapi.api.BasicKrakenExAPIPrivateWebsocketMethods*, *krakenexapi.api.RawCallRateLimitedKrakenExAPI*

Basic Kraken Exchange API, public + private endpoints.

Public Endpoints

class krakenexapi.api.**BasicKrakenExAPIPublicMethods**

Public Kraken Exchange API endpoints.

Most methods have additional post-processing, like conversion of strings to `float` (easier computation), or wrapping into NamedTuples to allow better access.

Raw responses can be retrieved via _ + method name. Alternatively, the *RawKrakenExAPI.query_public()* can be used.

Notes

See official API documentation (public)

_get_server_time() → Dict[str, Any]

get_server_time() → datetime.datetime

get_system_status()

get_asset_info(*asset: Optional[Union[str, List[str]]] = None*) → Dict[str, Dict[str, Any]]

_get_asset_pairs(*pair: Optional[Union[str, List[str]]] = None*, *info: Optional[str] = None*) → Dict[str, Dict[str, Any]]

_get_asset_pairs_static_values() → Dict[str, Union[int, float, str]]

get_asset_pairs(*pair: Optional[Union[str, List[str]]] = None*, *info: Optional[str] = None*) → Dict[str, Dict[str, Any]]

_get_ticker_information(*pair: Union[str, List[str]]*) → Dict[str, Dict[str, Any]]

get_ticker_information(*pair: Union[str, List[str]]*) → Dict[str, Dict[str, Any]]

_get_ohlc_data(*pair: str*, *interval: Optional[int] = None*, *since: Optional[int] = None*) → Tuple[List[List[Any]], int]

get_ohlc_data(*pair: str*, *interval: Optional[int] = None*, *since: Optional[int] = None*) → Tuple[List[List[Any]], int]

_get_order_book(*pair: str*, *count: Optional[int] = None*) → Tuple[List, List]

get_order_book(*pair: str*, *count: Optional[int] = None*) → Tuple[List, List]

_get_recent_trades(*pair: str*, *since: Optional[str] = None*) → Tuple[List[List[Any]], int]

get_recent_trades(*pair: str*, *since: Optional[str] = None*) → Tuple[List[List[Any]], int]

_get_recent_spread_data(*pair: str*, *since: Optional[int] = None*) → Tuple[List[List[Any]], int]

get_recent_spread_data(*pair: str*, *since: Optional[int] = None*) → Tuple[List[List[Any]], int]

Private Endpoints

class krakenexapi.api.**BasicKrakenExAPIPrivateUserDataMethods**

Private Kraken Exchange API user data endpoints.

Endpoints to retrieve:

- orders (open/closed),
- transactions,
- ledger entries,
- account/trade balance, volume (fee information)

Methods with `_info` suffix allow retrieval of information by IDs, others will return sliced subsets with a *total*/*offset*.

Notes

See official API documentation (user data)

get_account_balance() → Dict[str, float]

get_trade_balance(*asset: Optional[str] = None*) → Dict[str, float]

get_open_orders(*trades: Optional[bool] = None*, *userref: Optional[str] = None*) → Dict[str, Dict[str, Any]]

get_closed_orders(*trades: Optional[bool] = None*, *userref: Optional[str] = None*, *start: Optional[Union[int, float, str]] = None*, *end: Optional[Union[int, float, str]] = None*, *offset: Optional[int] = None*, *closetime: Optional[str] = None*) → Tuple[Dict[str, Dict[str, Any]], int]

get_orders_info(*txid: Union[str, List[str]]*, *trades: Optional[bool] = None*, *userref: Optional[str] = None*) → Dict[str, Dict[str, Any]]

get_trades_history(*type: Optional[str] = None*, *trades: Optional[bool] = None*, *start: Optional[Union[int, float, str]] = None*, *end: Optional[Union[int, float, str]] = None*, *offset: Optional[int] = None*) → Tuple[Dict[str, Dict[str, Any]], int]

get_trades_info(*txid: Union[str, List[str]]*, *trades: Optional[bool] = None*) → Dict[str, Dict[str, Any]]

get_open_positions(*txid: Union[str, List[str]]*, *docalcs: Optional[bool] = None*, *trades: Optional[bool] = None*, *consolidation: Optional[str] = None*) → Dict[str, Dict[str, Any]]

get_ledgers(*asset: Optional[Union[str, List[str]]] = None*, *type: Optional[str] = None*, *start: Optional[Union[int, float, str]] = None*, *end: Optional[Union[int, float, str]] = None*, *offset: Optional[int] = None*) → Tuple[Dict[str, Dict[str, Any]], int]

get_ledgers_info(*lid: Union[str, List[str]]*) → Dict[str, Dict[str, Any]]

get_trade_volume(*pair: Optional[Union[str, List[str]]] = None*, *fee_info: Optional[bool] = None*) → Dict[str, Any]

class krakenexapi.api.**BasicKrakenExAPIPrivateUserTradingMethods**

Private Kraken Exchange API user trading endpoints.

Notes

See official API documentation (user trading)

class krakenexapi.api.**BasicKrakenExAPIPrivateUserFundingMethods**
Private Kraken Exchange API user funding endpoints.

Notes

See official API documentation (user funding)

class krakenexapi.api.**BasicKrakenExAPIPrivateWebsocketMethods**
Private Kraken Exchange API websocket endpoint.

Notes

See official API documentation (websocket)

get_websocket_token() → str

4.2.3 Utility functions

To ease the gathering of complete lists of orders/trades/ledger entries. *The Kraken API will for some endpoints with possibly a large amount of entries split the response into chunks of 50 (or similar) and subsequent calls can use the* `ofs` (*offset* parameter) *and the returned* `total` *to gather all entries as needed. Note, that for some endpoints and argument choices the total will not be correct and the endpoint will return an empty dictionary instead. (which the functions below handle for you)*

krakenexapi.api.**gather_closed_orders**(*api:* krakenexapi.api.BasicKrakenExAPIPrivateUserDataMethods, **args*, ***kwargs*) → Dict[str, Any]
Gather a complete list of closed orders.

Wraps *get_closed_orders()* and iteratively queries next subsets until everything retrieved.

Parameters **api** (*BasicKrakenExAPIPrivateUserDataMethods*) – An API instance with access to private user data endpoints

Returns *Dict[str, Any]* – same as *get_closed_orders()*

krakenexapi.api.**gather_ledgers**(*api:* krakenexapi.api.BasicKrakenExAPIPrivateUserDataMethods, **args*, ***kwargs*) → Dict[str, Any]
Gather a complete list of ledger entries.

Wraps *get_ledgers()* and iteratively queries next subsets until everything retrieved.

Note, that *get_ledgers()* returns incorrect `total` if only parameterized with *type*, which will be handled here.

Parameters **api** (*BasicKrakenExAPIPrivateUserDataMethods*) – An API instance with access to private user data endpoints

Returns *Dict[str, Any]* – same as *get_ledgers()*

krakenexapi.api.**gather_trades**(*api:* krakenexapi.api.BasicKrakenExAPIPrivateUserDataMethods, **args*, ***kwargs*) → Dict[str, Any]
Gather a complete list of transactions (trades).

Wraps *get_trades_history()* and iteratively queries next subsets until everything retrieved.

Parameters **api** (*BasicKrakenExAPIPrivateUserDataMethods*) – An API instance with access to private user data endpoints

Returns *Dict[str, Any]* – same as `get_trades_history()`

4.2.4 Call Rate Limiting

The Kraken Exchange used different quotas for its API methods, see the article What are the API rate limits?.

class krakenexapi.api.**_CallRateLimitInfo**(*limit: float = 1*, *cost: float = 1*, *decay: float = 1.0*)

decay()

time_to_call(*cost: float*) → float

can_call(*cost: Optional[Union[int, float]] = None*) → bool

set_exceeded()

check(*cost: Optional[Union[int, float]] = None*) → bool

check_and_wait(*cost: Optional[Union[int, float]] = None*)

class krakenexapi.api.**KrakenExAPICallRateLimiter**(*tier: Optional[str] = None*)

_reset_account_crl()

static _is_private(*method: str*)

static _get_cost(*method: str*) → int

check_call(*method: str*, *wait: bool = True*) → bool

set_exceeded(*method*)

class krakenexapi.api.**RawCallRateLimitedKrakenExAPI**(*key: Optional[str] = None*, *secret: Optional[str] = None*, *tier: Optional[str] = 'Starter'*)

Extend `RawKrakenExAPI` with call rate limiting and request retry mechanisms.

Parameters **tier** (*str, optional*) – Kraken verification level, can be "Starter", "Intermediate", "Pro", by default "Starter"

Variables **_num_retries** (*int*) – Maximum number of retries, by default 3

_query_raw(*path: str*, *data: Dict[str, Any]*, *headers: Dict[str, Any]*, *timeout: Optional[Tuple[int, float]] = None*) → Dict[str, Any]

query_public(*method: str*, ***kwargs*) → Dict[str, Any]

query_private(*method: str*, *otp: Optional[str] = None*, ***kwargs*) → Dict[str, Any]

4.2.5 Exceptions

See *krakenexapi.exceptions*.

4.3 krakenexapi.exceptions

exception krakenexapi.exceptions.**KrakenExAPIError**
Bases: Exception

Generic error.

exception krakenexapi.exceptions.**APIRateLimitExceeded**
Bases: *krakenexapi.exceptions.KrakenExAPIError*

API Error: EAPI:Rate limit exceeded.

exception krakenexapi.exceptions.**APIArgumentUsageError**
Bases: *krakenexapi.exceptions.KrakenExAPIError*, ValueError

Error from Kraken API if arguments incorrectly supplied or used or values not supported.

API Error: EGeneral:Invalid arguments

exception krakenexapi.exceptions.**NoPrivateKey**
Bases: *krakenexapi.exceptions.KrakenExAPIError*

Thrown if trying to use a private Kraken Exchange API without a private key.

exception krakenexapi.exceptions.**NoSuchAPIMethod**
Bases: *krakenexapi.exceptions.KrakenExAPIError*

Error thrown if trying to use an invalid API method.

exception krakenexapi.exceptions.**APIPermissionDenied**
Bases: *krakenexapi.exceptions.KrakenExAPIError*

Error when trying to call API methods without given permission.

API Error: EGeneral:Permission denied

exception krakenexapi.exceptions.**APIInvalidNonce**
Bases: *krakenexapi.exceptions.KrakenExAPIError*

Error when trying to call API methods concurrently or out of order. Nonce will no be monotonic, so API throws error.

API Error: EAPI:Invalid nonce

4.4 krakenexapi.wallet

4.4.1 Currency

They are intended to work as singletons, and therefore track of registered/created instances.

class krakenexapi.wallet.**Currency**(*symbol: str*, *name: str*, *decimals: int*, *display_decimals: int*, *letter: Optional[str] = None*, *description: Optional[str] = None*)
Immutable singleton currency info dataclass.

Raises **AssertionError** – If trying to create a new *Currency singleton* instance with an already registered *symbol* identifier.

symbol: str
Currency symbol used by Kraken Exchange.

name: str
Alternative currency symbol. (can be same as *symbol*, often without *X*/*Z* prefix)

decimals: int
Number of decimals used in computation (precision).

display_decimals: int
Number of decimals displayed to user.

letter: Optional[str] = None
Optional. Currency symbol/glyph.

description: Optional[str] = None
Optional. Short currency description/name.

property is_fiat
Returns `True` if currency is fiat, `False` if crypto.
Returns *bool* – `True` if fiat currency.

property is_staked_onchain
On-chain staked currency.
Returns *bool*

property is_staked_offchain
Off-chain staked currency.
Returns *bool*

property is_staked
Is a currency, staked on Kraken Exchange.
Returns *bool*

format_value(*value: float*) → str
Formats a given *value* according to the *display_decimals* of the currency.

If a currency symbol/letter exists, append it.
Parameters **value** (*float*) – The value to be formatted.
Returns *str* – Formatted string.

round_value(*value: float*) → float
Round a given *value* to the maximum number of digits as specified in *decimals* of the currency.
Parameters **value** (*float*) – Number to be rounded.
Returns *float* – Number rounded to *decimals* digits.

classmethod find(*symbol: str*) → *krakenexapi.wallet.Currency*
Finds the singleton instance of the currency described by the *symbol* (Kraken Exchange identifier) string.
Returns *krakenexapi.wallet.Currency* – Currency singleton instance.
Raises **KeyError** – If no *Currency* exists for the *symbol*.

classmethod all_symbols(*unique: bool = True*) → Set[str]
Return a list of all instanciated *Currency* symbols.

This allows the retrieval of all *Currency* instances via `find()` (with/without) duplicates.
Parameters **unique** (*bool, optional*) – Whethere the list of symbols allows for duplicate when used for retrieval, or not, by default True (no duplicates)
Returns *Set[str]* – Currency symbol names (used on Kraken Exchange)

classmethod build_from_api(*api:* krakenexapi.api.BasicKrakenExAPIPublicMethods)
Uses the *api* to query a list of all currencies on Kraken Exchange and builds singleton instances for each of it.

Parameters **api** (*BasicKrakenExAPIPublicMethods*) – An API that allows to query public Kraken Exchange endpoints.

class krakenexapi.wallet.**CurrencyPair**(*symbol: str*, *altname: str*, *name: str*, *pair_decimals: int*, *base:* krakenexapi.wallet.Currency, *quote:* krakenexapi.wallet.Currency, *ordermin: Optional[float] = None*)
Immutable singleton currency trading pair info dataclass.

Raises **AssertionError** – If trying to create a new *CurrencyPair singleton* instance with an already registered *symbol* identifier.

symbol: str
Trading pair symbol (used on Kraken Exchange)

altname: str
Alternative name for trading pair.

name: str
Visual name (human readable).

base: *krakenexapi.wallet.Currency*
Base currency

quote: *krakenexapi.wallet.Currency*
Quote currency, determines value/price for *base* currency.

ordermin: Optional[float] = None
Minimum amount of *base* currency for a new order.

property is_fiat2crypto
Trading pair between crypto and fiat currency.

Returns *bool*

property is_crypto2crypto
Trading pair between two crypto currencies.

Returns *bool*

property is_fiat2fiat
Trading pair between two fiat currencies.

Returns *bool*

classmethod find(*symbol: str*) → *krakenexapi.wallet.CurrencyPair*
Find the singleton *CurrencyPair* instance for the given *symbol* names. (as used on Kraken Exchange)

Returns *krakenexapi.wallet.CurrencyPair* – The *CurrencyPair* singleton instance.

Raises **KeyError** – If no *CurrencyPair* exists for the *symbol*.

classmethod all_symbols(*unique: bool = True*) → Set[str]
Return a list of *symbol* for all registered *CurrencyPair* instances.

Parameters **unique** (*bool, optional*) – Whether the list should contain all registered *symbol* and *name*, or just enough to allow retrieval of all singleton instances, by default True

Returns *Set[str]* – List of currency pair symbols. (used on Kraken Exchange)

classmethod build_from_api(*api:* krakenexapi.api.BasicKrakenExAPIPublicMethods)
Build a list of *CurrencyPair* and optionally *Currency* singleton instances.

Uses the public Kraken Exchange API to retrieve a list of all available currency trading pairs and build instances for each of it.

Parameters **api** (*BasicKrakenExAPIPublicMethods*) – An API object that allows querying the public Kraken Exchange endpoints.

4.4.2 Transactions

class krakenexapi.wallet.**TradingTransaction**(*currency_pair: krakenexapi.wallet.CurrencyPair*, *price: float*, *amount: float*, *cost: float*, *fees: float*, *timestamp: datetime.datetime*, *txid: str*, *otxid: Optional[Union[float, str]] = None*)

A trading transaction.

Subclasses should be used to mark *buy* / *sell* type.

currency_pair: *krakenexapi.wallet.CurrencyPair*
Currency pair, of base and quote currency. *quote* currency determines the price. *base* is the currency being traded.

price: float
Price of (crypto) currency.

amount: float
Amount of currency.

cost: float
Cost of *base* currency (*currency_pair*) in *quote* currency.

fees: float
Fees for transactions, in *quote* currency.

timestamp: datetime.datetime
Timestamp of transaction.

txid: str
Transaction ID.

otxid: Optional[Union[float, str]] = None
Optional. Order ID associated with transaction.

property base_currency
Base currency being traded.
Returns *Currency*

property quote_currency
Quote currency. Determines the price and value of the *base_currency* currency.
Returns *Currency*

class krakenexapi.wallet.**CryptoBuyTransaction**(*currency_pair: krakenexapi.wallet.CurrencyPair*, *price: float*, *amount: float*, *cost: float*, *fees: float*, *timestamp: datetime.datetime*, *txid: str*, *otxid: Optional[Union[float, str]] = None*)

Bases: *krakenexapi.wallet.TradingTransaction*

Crypto currency buy transaction.

class krakenexapi.wallet.**CryptoSellTransaction**(*currency_pair: krakenexapi.wallet.CurrencyPair, price: float, amount: float, cost: float, fees: float, timestamp: datetime.datetime, txid: str, otxid: Optional[Union[float, str]] = None*)

Bases: *krakenexapi.wallet.TradingTransaction*

Crypto currency sell transaction.

class krakenexapi.wallet.**FundingTransaction**(*currency: krakenexapi.wallet.Currency, amount: float, timestamp: datetime.datetime, fees: float = 0.0, lxid: Optional[Union[int, float, str]] = None*)

A funding transaction.

Subclasses show whether it is a `deposit` or `withdrawal`.

currency: *krakenexapi.wallet.Currency*
Currency.

amount: float
Amount of currency in transaction.

timestamp: datetime.datetime
Timestamp of transaction.

fees: float = 0.0
Fees for transaction.

lxid: Optional[Union[int, float, str]] = None
Ledger ID. (more exact than timestamp if used in API queries)

class krakenexapi.wallet.**DepositTransaction**(*currency: krakenexapi.wallet.Currency, amount: float, timestamp: datetime.datetime, fees: float = 0.0, lxid: Optional[Union[int, float, str]] = None*)

Bases: *krakenexapi.wallet.FundingTransaction*

Deposit transaction of (fiat) currency to Kraken Exchange.

class krakenexapi.wallet.**WithdrawalTransaction**(*currency: krakenexapi.wallet.Currency, amount: float, timestamp: datetime.datetime, fees: float = 0.0, lxid: Optional[Union[int, float, str]] = None*)

Bases: *krakenexapi.wallet.FundingTransaction*

Withdrawal transaction from the Kraken Exchange.

4.4.3 Assets and Wallet

class krakenexapi.wallet.**Asset**(*currency: krakenexapi.wallet.Currency, api: krakenexapi.api.BasicKrakenExAPIPrivateUserDataMethods, quote_currency: Optional[krakenexapi.wallet.Currency] = None*)

Wrapper around a *Currency* and *api* for easy asset win/loss computation.

property has_transactions
Return `True` if the *currency* has transactions.
Returns *bool* – `True` if transactions exist.

property currency
The asset currency.
Returns *Currency*

property amount
Amount of *currency* the user has on Kraken Exchange since the last update.
Returns *float* – Amount of currency.

property amount_buy
Total amount of *currency* being bought.

Retrieved from API via transactions.
Returns *float* – total amount of currency bought.

property amount_sell
Total amount of *currency* being sold.

Retrieved from API via transactions.
Returns *float* – total amount of currency sold.

property amount_by_transactions
Total of bought (*amount_buy*) and sold (*amount_sell*) amount.
Returns *float* – Amount of currency just by buy/sell transactions.

Notes

$$amount = amount_{buy} - amount_{sell}$$

property price_buy_avg
Average price of all *buy* transactions.
Returns *float* – average buy price

Notes

Price calculation

$$price_{buy} = \frac{\sum cost_{buy}}{\sum amount_{buy}}$$

property price_sell_avg
Average price of all *sell* transactions.
Returns *float* – average sell price
See also:

price_buy_avg

property price_avg
Average price based on *cost* by transaction and current *amount*.
Returns *float* – average price
See also:

cost, *amount*

price_for_noloss (*fees_sell: float = 0.26*) → float
Minimum price that should be used for *sell* if no loss should be incurred. Prices larger than the noloss price will be wins.
Parameters **fees_sell** (*float, optional*) – Kraken Exchange *sell* fees, by default 0.26 (maximum for market orders)

Returns *float* – noloss price (based on quote currency)

Note: Current computation may not be completely correct but should be rather close. **Please** verify manually! (e.g. adjust price ±5%)

$$price_{noloss} = price_{avg} * \frac{1 + fees_{total}}{1 - fees_{sell}}$$

with $fees_{total}$ being all fees incurred through transactions

---

See also:

price_avg, *fees_percentage*

property cost_buy
Sum of costs of *buy* transactions, based on *quote* currency.
Returns *float* – total costs of buy transactions (without fees)

property cost_sell
Sum of costs of *sell* transactions, based on *quote* currency.
Returns *float* – total costs of sell transactions (without fees)

property cost
Total costs computed by *cost_buy* and *cost_sell*.

So, only costs computed by transactions. Negative costs means that costs of *buy* is higher than *sell*, positive costs means the opposite. (Positive costs would mean a win based on transactions alone.)
Returns *float* – Difference of costs for *buy* and *sell*

Notes

$$cost = -cost_{buy} + cost_{sell}$$

See also:

cost_buy, *cost_sell*, *fees*

property fees_buy
Fees incurred by *buy* transactions.
Returns *float* – total sum of fees

property fees_sell
Fees incurred by *sell* transactions.
Returns *float* – total sum of fees

property fees
Total sum of fees for both *buy* and *sell* transactions.
Returns *float* – total sum of fees
See also:

fees_buy, *fees_sell*, *fees_percentage*

property fees_percentage
Percentage of fees compared to costs, of transactions.
Returns *float* – fee percentage, see Kraken Exchange fees

Notes

$$fees_{\%} = 100 * \frac{fees}{cost_{buy} + cost_{sell}}$$

property is_loss
Loss based on transactions. Loss if costs for *buy* higher than *sell*.
Returns *bool* – `True` if loss (i.e. higher costs for *buy*)

class krakenexapi.wallet.**Fund**(*currency:* krakenexapi.wallet.Currency, *api:* krakenexapi.api.BasicKrakenExAPIPrivateUserDataMethods)
Wrapper around fiat *Currency* info and computations.

property has_transactions
Return `True` if the *currency* has ledger entries for *deposit* and *withdrawal*.
Returns *bool* – `True` if transactions exist.

property currency
The fund fiat currency.
Returns *Currency*

property amount
Amount of *currency* the user has on Kraken Exchange since the last update.
Returns *float* – Amount of currency.

property amount_deposit
Total amount of *currency* being deposited.

Retrieved from API via ledger entries.
Returns *float* – total amount of currency deposited.

property amount_withdrawal
Total amount of *currency* being withdrawn.

Retrieved from API via ledger entries.
Returns *float* – total amount of currency withdrawn.

property amount_by_transactions
Total of deposited (*amount_deposit*) and withdrawn (*amount_withdrawal*) amount.
Returns *float* – Amount of currency just by deposit/withdrawal transactions.

Notes

$$amount = amount_{deposit} - amount_{withdrawal}$$

property fees_deposit
Fees incurred by *deposit* transactions (ledgers).
Returns *float* – total sum of fees

property fees_withdrawal
Fees incurred by *withdrawal* transactions (ledgers).
Returns *float* – total sum of fees

property fees
Total sum of fees for both *deposit* and *withdrawal* transactions.
Returns *float* – total sum of fees
See also:

fees_deposit, *fees_withdrawal*, *fees_percentage*

property fees_percentage
Percentage of fees compared to amount of fiat currency.
Returns *float* – fee percentage

Notes

$$fees_{\%} = 100 * \frac{fees}{amount_{deposit} + amount_{withdrawal}}$$

class krakenexapi.wallet.**Wallet**(*api:* krakenexapi.api.BasicKrakenExAPI)

_update()
Update assets and funds.

_update_assets()
Update internal dictionary of *Asset*.

If no assets exist, create an initial dictionary of assets.

Check if new transactions found, then update all assets. If *_last_txid* is not None then query a subset of trading transactions to reduce traffic/calls.

_update_funds()
Update internal dictionary of *Fund*.

If no assets exist, create an initial dictionary of funds.

Check if new funding transactions found, then update all funds. If *_last_lxid* is not None then query a subset of funding transactions to reduce traffic/calls.

static get_all_account_currencies(*api:* krakenexapi.api.BasicKrakenExAPIPrivateUserDataMethods) → List[*krakenexapi.wallet.Currency*]
Retrieve a full list of currencies used on Kraken Exchange by the user.

Current balance might not even show currency because sold, withdrawn or staked etc.
1. Will look for currencies by trading transactions,
2. then will look at deposit/withdrawals ledger entries,
3. **WIP** then staking/transfering ledger entries,
4. Currencies listed in current account balance.

Parameters **api** (*BasicKrakenExAPIPrivateUserDataMethods*) – An API object that allows querying private endpoints
Returns *List[Currency]* – List of all Currencies used by trader.

See also:

get_account_crypto_currencies, *get_account_fiat_currencies*

static get_account_crypto_currencies(*api:* krakenexapi.api.BasicKrakenExAPIPrivateUserDataMethods) → List[*krakenexapi.wallet.Currency*]
Return a list of crypto currencies used by the trader.
Parameters **api** (*BasicKrakenExAPIPrivateUserDataMethods*) – An API instance with access to private Kraken Exchange endpoints.
Returns *List[Currency]* – List of crypto currencies

static get_account_fiat_currencies(*api:* krakenexapi.api.BasicKrakenExAPIPrivateUserDataMethods) → List[*krakenexapi.wallet.Currency*]
Return a list of fiat currencies used by the trader.
Parameters **api** (*BasicKrakenExAPIPrivateUserDataMethods*) – An API instance with access to private Kraken Exchange endpoints.

Returns *List[Currency]* – List of fiat currencies

static build_assets_from_api(*api:* krakenexapi.api.BasicKrakenExAPIPrivateUserDataMethods, *fiat_currency:* *Optional[*krakenexapi.wallet.Currency*]* = *None*) → List[*krakenexapi.wallet.Asset*]

Build a list of `Asset` from the list of crypto currencies of the trader.

Optionally set a quote currency for computations.

Parameters

- **api** (*BasicKrakenExAPIPrivateUserDataMethods*) – An API instance with access to private Kraken Exchange endpoints.
- **fiat_currency** (*Optional[Currency], optional*) – Fiat currency for asset value computations, by default None

Returns *List[Asset]* – List of crypto currency assets.

static build_funds_from_api(*api:* krakenexapi.api.BasicKrakenExAPIPrivateUserDataMethods)

Build a list of `Fund` from the list of fiat currencies of the trader.

Parameters **api** (*BasicKrakenExAPIPrivateUserDataMethods*) – An API instance with access to private Kraken Exchange endpoints.

Returns *List[Fund]* – List of fiat currency funds.

static build_funding_transactions(*api:* krakenexapi.api.BasicKrakenExAPIPrivateUserDataMethods, *start:* *Optional[Union[int, float, str]]* = *None*, *sort:* *bool* = *True*) → List[*krakenexapi.wallet.FundingTransaction*]

Build a list of funding transactions based on ledger entries.

Parameters

- **api** (*BasicKrakenExAPIPrivateUserDataMethods*) – An API instance to query private Kraken Exchange endpoints
- **start** (*Optional[Union[int, float, str]], optional*) – Start timestamp or ledger LXID, by default None
- **sort** (*bool, optional*) – Whether to sort transactions by timestamp ascending, by default True

Returns *List[FundingTransaction]* – List of funding transactions

Raises **RuntimeError** – On unknown funding (*deposit/withdrawal*) transaction type.

static build_trading_transactions(*api:* krakenexapi.api.BasicKrakenExAPIPrivateUserDataMethods, *start:* *Optional[Union[int, float, str]]* = *None*, *sort:* *bool* = *True*) → List[*krakenexapi.wallet.TradingTransaction*]

Build a list of trading transactions.

Parameters

- **api** (*BasicKrakenExAPIPrivateUserDataMethods*) – An API instance to query private Kraken Exchange endpoints.
- **start** (*Optional[Union[int, float, str]], optional*) – Start unix timestamp or transaction TXID, by default None
- **sort** (*bool, optional*) – Whether to sort transactions by timestamp ascending, by default True

Returns *List[TradingTransaction]* – List of trading transactions

Raises **RuntimeError** – On unknown transaction type.

CHAPTER

FIVE

CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

5.1 Bug reports

When reporting a bug please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.2 Documentation improvements

KrakenExApi could always use more documentation, whether as part of the official KrakenExApi docs, in docstrings, or even on the web in blog posts, articles, and such.

5.3 Feature requests and feedback

The best way to send feedback is to file an issue at https://github.com/Querela/python-krakenexapi/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that code contributions are welcome :)

5.4 Development

To set up *python-krakenexapi* for local development:

1. Fork python-krakenexapi (look for the "Fork" button).
2. Clone your fork locally:

```
git clone git@github.com:YOURGITHUBNAME/python-krakenexapi.git
```

3. Create a branch for local development:

```
git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

4. When you're done making changes run all the checks and docs builder with tox one command:

```
tox
```

5. Commit your changes and push your branch to GitHub:

```
git add .
git commit -m "Your detailed description of your changes."
git push origin name-of-your-bugfix-or-feature
```

6. Submit a pull request through the GitHub website.

5.4.1 License

Any contribution that you make to this repository will be under the MIT license.

5.4.2 Developer Certificate of Origin

Contributors must sign-off each commit by adding a *Signed-off-by: ...* line to commit messages to certify that they have the right to submit the code they are contributing to the project according to the Developer Certificate of Origin (DCO).

5.4.3 Pull Request Guidelines

If you need some code review or feedback while you're developing the code just make the pull request.

For merging, you should:

1. Include passing tests (run `tox`)[1].
2. Update documentation when there's new API, functionality etc.
3. Add a note to `CHANGELOG.rst` about the changes.
4. Add yourself to `AUTHORS.rst`.

5.4.4 Tips

To run a subset of tests:

```
tox -e envname -- pytest -k test_myfeature
```

To run all the test environments in *parallel*:

```
tox -p auto
```

CHAPTER

SIX

AUTHORS

- Erik Körner - ekdev@live.de

CHAPTER

SEVEN

MIT LICENSE

CHAPTER

EIGHT

CHANGELOG

8.1 WIP

- Add tests (raw query, public + private API).
- Add tooling for docs, checks.
- Add meta info dataclasses. . .
- Fix many minor issues, badges, docs, workflows/tests.
- Add wrappers around transactions, handling, statistics etc.
- Rename private user data ledgers endpoints!
- Extract exceptions into own module.
- Start with websocket API.
- Add live-api tests (markers, github workflow).
- Call rate limiting - retry with backoff.
- Add Funds (same as Assets) to `wallet.py`.
- Rewrite `wallet.py`, group functions, add *`Wallet`*.

8.2 0.0.1 (2020-12-27)

- Add raw kraken Exchange API.
- Add basic Kraken Exchange API - public methods.
- Add custom exceptions.
- Add basic response classes for better accessing of entries, fix stringified floats.
- Add call rate limiting (crl).
- Add tests for crl, more in work.
- Add private user data API endpoints.
- Add basic documentation.
- First official pre-release.

8.3 0.0.0 (2020-12-25)

- First release on PyPI.

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

k

INDEX

Symbols

A

B

C

D

F

G

H

I

K

L