python-kraken-sdk

Benjamin Thomas Schwertfeger

Apr 18, 2026

CONTENTS:

CHAPTER

ONE

PYTHON-KRAKEN-SDK

This is the documentation of the unofficial Python SDK to interact with the Kraken Crypto Asset Exchange.

Payward Ltd. and Kraken are in no way associated with the authors of this package and documentation. Please note that this project is independent and not endorsed by Kraken or Payward Ltd. Users should be aware that they are using third-party software, and the authors of this project are not responsible for any issues, losses, or risks associated with its usage.

This documentation refers to the python-kraken-sdk and serves to simplify the application of trading strategies, in which as far as possible all interaction possibilities with the crypto asset exchange Kraken are implemented, tested and documented.

- Gladly open an issue on GitHub on make if something is incorrect or missing (python-kraken-sdk/issues).
- The output in the examples may differ, as these are only intended as examples and may change in the future.
- If a certain endpoint is not reachable, the function `kraken.spot.SpotClient.request()` or `kraken.futures.FuturesClient.request()`, which is also available in all derived REST clients, can be used to reach an endpoint with the appropriate parameters. Here private content can also be accessed, provided that either the base class or one of the clients has been initialized with valid credentials.
- For Futures there is the websocket client *kraken.futures.FuturesWSClient* and for Spot and xStocks *kraken.spot.SpotWSClient*.

1.1 Disclaimer

There is no guarantee that this software will work flawlessly at this or later times. Of course, no responsibility is taken for possible profits or losses. This software probably has some errors in it, so use it at your own risk. Also no one should be motivated or tempted to invest assets in speculative forms of investment. By using this software you release the author(s) from any liability regarding the use of this software.

1.2 Features

General:

- Command-line interface
- Access both public and private, REST and websocket endpoints
- Responsive error handling and custom exceptions
- Extensive examples
- Tested using the pytest framework

- Releases are permanently archived at Zenodo

Available Clients:

- Spot REST Clients - including xStocks capability
- Spot Websocket Client (Websocket API v2)
- Spot Orderbook Client (Websocket API v2)
- Futures REST Clients
- Futures Websocket Client

Projects using this SDK:

- https://github.com/btschwertfeger/infinity-grid
- https://github.com/btschwertfeger/kraken-rebalance-bot
- https://github.com/btschwertfeger/python-kraken-sdk/network/dependents

1.3 Important Notice

ONLY tagged releases are available at PyPI. The content of the master branch may not match with the content of the latest release. - Please have a look at the release specific READMEs and changelogs.

It is also recommended to **pin the used version** to avoid unexpected behavior on new releases.

1.4 Troubleshooting

- Check if you downloaded and installed the **latest version** of the python-kraken-sdk.
- Check the **permissions of your API keys** and the required permissions on the respective endpoints.
- If you get some Cloudflare or **rate limit errors**, please check your Kraken Tier level and maybe apply for a higher rank if required.
- **Use different API keys for different algorithms**, because the nonce calculation is based on timestamps and a sent nonce must always be the highest nonce ever sent of that API key. Having multiple algorithms using the same keys will result in invalid nonce errors.
- Always keep an eye on https://status.kraken.com/ when encountering connectivity problems.
- Feel free to open an issue at python-kraken-sdk/issues.
- The xStocks feature is not available globally. Please checkout Kraken's documentation to get to know the availability zones.

1.5 References

- https://python-kraken-sdk.readthedocs.io/en/stable
- https://docs.kraken.com/api/
- https://docs.kraken.com/api/docs/guides/global-intro
- https://support.kraken.com/hc/en-us/sections/360012894412-Futures-API

CHAPTER

TWO

GETTING STARTED

1. Install the Python module

```
python3 -m pip install python-kraken-sdk
```

2. Register at Kraken and generate API Keys

- Spot Trading: https://pro.kraken.com/app/settings/api
- Futures Trading: https://futures.kraken.com/trade/settings/api
- Futures Sandbox: https://demo-futures.kraken.com/settings/api

3. Start using the provided example scripts

Examples can be found within the repository of the python-kraken-sdk and obviously in this documentation. See the *Usage Examples* section for basic usage examples and *Trading Bot Templates* to get impressed by orderbook clients, as well as boilerplates for trading bots using the SDK.

4. Error handling

If any unexpected behavior occurs, please check **your API permissions**, **rate limits**, update the python-kraken-sdk, see the *Troubleshooting* section, and if the error persists please open an issue at python-kraken-sdk/issues.

CHAPTER

THREE

COMMAND-LINE INTERFACE

The python-kraken-sdk provides a command-line interface to access the Kraken API using basic instructions while performing authentication tasks in the background. The Spot and Futures APIs are accessible and follow the pattern `kraken {spot,futures} [OPTIONS] URL`. All endpoints of the Kraken Spot and Futurs API can be accessed like that. See examples below.

Listing 1: Command-line Interface Examples

```
# get server time
kraken spot https://api.kraken.com/0/public/Time
{'unixtime': 1716707589, 'rfc1123': 'Sun, 26 May 24 07:13:09 +0000'}

# get user's balances
kraken spot --api-key=<api-key> --secret-key=<secret-key> -X POST https://api.kraken.com/
↪0/private/Balance
{'ATOM': '17.28229999', 'BCH': '0.0000077100', 'ZUSD': '1000.0000'}

# get user's trade balances
kraken spot --api-key=<api-key> --secret-key=<secret-key> -X POST https://api.kraken.com/
↪0/private/TradeBalance --data '{"asset": "DOT"}'
{'eb': '2.8987347115', 'tb': '1.1694303513', 'm': '0.0000000000', 'uv': '0', 'n': '0.
↪0000000000', 'c': '0.0000000000', 'v': '0.0000000000', 'e': '1.1694303513', 'mf': '1.
↪1694303513'}

# get 1D candles for a futures instrument
kraken futures https://futures.kraken.com/api/charts/v1/spot/PI_XBTUSD/1d
{'candles': [{'time': 1625616000000, 'open': '34557.840000000000', 'high': '34803.
↪20000000000', 'low': '33816.320000000000', 'close': '33880.220000000000', 'volume': '0' .
↪..

# get user's open futures positions
kraken futures --api-key=<api-key> --secret-key=<secret-key> https://futures.kraken.com/
↪derivatives/api/v3/openpositions
{'result': 'success', 'openPositions': [], 'serverTime': '2024-05-26T07:15:38.91Z'}
```

3.1 kraken

Command-line tool to access the Kraken Crypto Asset Exchange API

Usage

```
kraken [OPTIONS] COMMAND [ARGS]...
```

Options

--version

-v, --verbose

Increase verbosity

3.1.1 futures

Access the Kraken Futures REST API

Usage

```
kraken futures [OPTIONS] URL
```

Options

-X <x>

Required Request method

Default
'GET'

Options
GET | POST | PUT | DELETE

-d, --data <data>

POST parameters as valid JSON string

-q, --query <query>

Query parameters as valid JSON string

--timeout <timeout>

Timeout in seconds

--api-key <api_key>

Kraken Public API Key

--secret-key <secret_key>

Kraken Secret API Key

Arguments

URL

Required argument

3.1.2 spot

Access the Kraken Spot REST API

Usage

```
kraken spot [OPTIONS] URL
```

Options

-X <x>

Required Request method

Default
'GET'

Options
GET | POST | PUT | DELETE

-d, **--data** <data>

Payload as valid JSON string

--timeout <timeout>

Timeout in seconds

--api-key <api_key>

Kraken Public API Key

--secret-key <secret_key>

Kraken Secret API Key

Arguments

URL

Required argument

CHAPTER

FOUR

USAGE EXAMPLES

The python-kraken-sdk provides lots of functions to easily access most of the REST and websocket endpoints of the Kraken Crypto Asset Exchange API. Since these endpoints and their parameters may change, all implemented endpoints are tested on a regular basis.

The repository of the python-kraken-sdk provides some example scripts that demonstrate some of the implemented methods. Please see the sections listed below.

Projects that use the SDK are listed below:

- https://github.com/btschwertfeger/infinity-grid
- https://github.com/btschwertfeger/kraken-rebalance-bot

4.1 Spot REST

The examples presented below serve to demonstrate the usage of the Spot REST clients provided by python-kraken-sdk to access Kraken's REST API.

For questions, feedback, additions, suggestions for improvement or problems python-kraken-sdk/discussions or python-kraken-sdk/issues may be helpful.

See https://docs.kraken.com/api/docs/guides/global-intro for information about the available endpoints and their usage.

The Spot client provides access to all un-and authenticated endpoints of Kraken's Spot API.

Listing 1: Example: Spot Client Usage (1)

```
from kraken.spot import SpotClient

client = SpotClient(key="<your-api-key>", secret="<your-secret-key>")
print(client.request("POST", "/0/private/Balance"))
```

The async Spot client allows for asynchronous access to Kraken's Spot API endpoints. Below are two examples demonstrating its usage.

Using SpotAsyncClient without a context manager; In this example, the client is manually closed after the request is made.

Listing 2: Example: Spot Client Usage (2)

```
import asyncio
from kraken.spot import SpotAsyncClient

async def main():
```

(continues on next page)

(continued from previous page)

```
    client = SpotAsyncClient(key="<your-api-key>", secret="<your-secret-key>")
    try:
        response = await client.request("POST", "/0/private/Balance")
        print(response)
    finally:
        await client.async_close()

asyncio.run(main())
```

Using SpotAsyncClient as context manager; This example demonstrates the use of the context manager, which ensures the client is automatically closed after the request is completed.

Listing 3: Example: Spot Client Usage (3)

```
import asyncio
from kraken.spot import SpotAsyncClient

async def main():
    async with SpotAsyncClient(
        key="<your-api-key>", secret="<your-secret-key>"
    ) as client:
        response = await client.request("POST", "/0/private/Balance")
        print(response)

asyncio.run(main())
```

The following legacy examples are not maintained on a regular basis. They serve only for demonstration purposes - make sure to checkout the documentation of the individual functions.

Listing 4: Example usage of Spot REST clients

```
# !/usr/bin/env python3
# -*- mode: python; coding: utf-8 -*-
#
# Copyright (C) 2023 Benjamin Thomas Schwertfeger
# All rights reserved.
# https://github.com/btschwertfeger
#

"""
Module that implements *some* examples for the Kraken Spot REST clients usage.
"""

import logging
import os
import time
from pathlib import Path

from kraken.spot import Funding, Market, Trade, User

logging.basicConfig(
    format="%(asctime)s %(module)s,line: %(lineno)d %(levelname)8s | %(message)s",
    datefmt="%Y/%m/%d %H:%M:%S",
```

(continues on next page)

(continued from previous page)

```
    level=logging.INFO,
)
logging.getLogger("requests").setLevel(logging.WARNING)
logging.getLogger("urllib3").setLevel(logging.WARNING)


key = os.getenv("SPOT_API_KEY")
secret = os.getenv("SPOT_SECRET_KEY")


def user_examples() -> None:
    """Example usage of the Spot User client"""
    # Usage of the User client to access private endpoints:
    user = User(key=key, secret=secret)

    print(user.get_account_balance())
    print(user.get_trade_balance())  # asset="BTC"
    print(user.get_open_orders())
    print(user.get_closed_orders())
    print(
        user.get_orders_info(
            txid="OBQFM7-JNVKS-H3ULEH",  # or txid="id1,id2,id3" or txid=["id1","id2"]
        ),
    )
    print(user.get_trades_history())
    time.sleep(3)  # to avoid rate limit
    print(user.get_trades_info(txid="TCNTTR-QBEVO-E5H5UK"))
    print(user.get_open_positions())  # or txid="someid"
    print(
        user.get_ledgers_info(),  # asset="BTC" or asset="BTC,EUR" or asset=["BTC","EUR"]
    )
    print(user.get_ledgers(id_="LIORGR-33NXH-LBUS5Z"))
    print(user.get_trade_volume())  # pair="BTC/EUR"

    # Exporting a ledger and trade report can be useful for analysis or
    # record-keeping purposes:
    response = user.request_export_report(
        report="ledgers",  # or report="trades"
        description="myLedgers1",
        format_="CSV",
    )
    print(user.get_export_report_status(report="ledgers"))

    # save report to file
    response_data = user.retrieve_export(id_=response["id"])
    with Path("myExport.zip").open("wb") as file:
        for chunk in response_data.iter_content(chunk_size=512):
            if chunk:
                file.write(chunk)

    print(
        user.delete_export_report(id_=response["id"], type_="delete"),
```

(continues on next page)

(continued from previous page)

```
    )


def market_examples() -> None:
    """Example usage of the Spot Market client"""
    market = Market()

    print(market.get_assets(assets=["XBT"]))
    print(market.get_asset_pairs(pair=["DOTEUR"]))
    print(market.get_ticker(pair="XBTUSD"))
    print(market.get_ohlc(pair="XBTUSD", interval=5))
    print(market.get_order_book(pair="XBTUSD", count=10))
    print(market.get_recent_trades(pair="XBTUSD"))
    print(market.get_recent_spreads(pair="XBTUSD"))
    print(market.get_system_status())
    time.sleep(2)


def trade_examples() -> None:
    """Example usage of the Spot Trade client"""
    print(
        "Attention: Please check if you really want to execute trade functions."
        " Running them without caution may lead to unintended orders!",
    )
    return
    trade = Trade(key=key, secret=secret)

    print(
        trade.create_order(
            ordertype="limit",
            side="buy",
            volume=1,
            pair="BTC/EUR",
            price=0.01,
        ),
    )
    print(
        trade.create_order_batch(
            orders=[
                {
                    "close": {
                        "ordertype": "stop-loss-limit",
                        "price": 120,
                        "price2": 110,
                    },
                    "ordertype": "limit",
                    "price": 140,
                    "price2": 130,
                    "timeinforce": "GTC",
                    "type": "buy",
                    "userref": "345dsdfddfgdsgdfgsfdsfsdf",
                    "volume": 1000,
```

(continues on next page)

(continued from previous page)

```
                },
                {
                    "ordertype": "limit",
                    "price": 150,
                    "timeinforce": "GTC",
                    "type": "sell",
                    "userref": "1dfgesggwe5t3",
                    "volume": 123,
                },
            ],
            pair="BTC/USD",
            validate=True,
        ),
    )

    print(
        trade.edit_order(txid="sometxid", pair="BTC/EUR", volume=4.2, price=17000),
    )
    time.sleep(2)

    print(trade.cancel_order(txid="O2JLFP-VYFIW-35ZAAE"))
    print(trade.cancel_all_orders())
    print(trade.cancel_all_orders_after_x(timeout=6))

    print(
        trade.cancel_order_batch(
            orders=[
                "O2JLFP-VYFIW-35ZAAE",
                "O523KJ-DO4M2-KAT243",
                "OCDIAL-YC66C-DOF7HS",
                "OVFPZ2-DA2GV-VBFVVI",
            ],
        ),
    )


def funding_examples() -> None:
    """Example usage of the Funding client"""
    funding = Funding(key=key, secret=secret)
    print(funding.get_deposit_methods(asset="DOT"))
    # print(funding.get_deposit_address(asset="DOT", method="Polkadot"))
    # print(funding.get_recent_deposits_status(asset="DOT"))
    print(
        funding.get_withdrawal_info(asset="DOT", key="MyPolkadotWallet", amount="200"),
    )

    print(
        "Attention: Please check if you really want to execute funding functions."
        " Running them without caution may lead to unintended withdrawals!",
    )
    return
    time.sleep(2)  # to avoid rate limit
```

(continues on next page)

(continued from previous page)

```
    print(funding.withdraw_funds(asset="DOT", key="MyPolkadotWallet", amount=200))
    print(funding.get_recent_withdraw_status(asset="DOT"))
    print(funding.cancel_withdraw(asset="DOT", refid="12345"))
    print(
        funding.wallet_transfer(
            asset="ETH",
            amount=0.100,
            from_="Spot Wallet",
            to_="Futures Wallet",
        ),
    )


def main() -> None:
    """Uncomment the examples you want to run:"""
    # NOTE: These are only examples that show how to use the clients, there are
    #       many other functions available in the clients.

    # user_examples()
    # market_examples()
    # trade_examples()
    # funding_examples()


if __name__ == "__main__":
    main()
```

4.2 Spot Websocket

The examples presented below serve to demonstrate the usage of the Spot websocket clients provided by python-kraken-sdk to access Kraken's Websocket API v2.

For questions, feedback, additions, suggestions for improvement or problems python-kraken-sdk/discussions or python-kraken-sdk/issues may be helpful.

Listing 5: Example access and usage for Kraken Spot Websocket API

```
# !/usr/bin/env python3
# -*- mode: python; coding: utf-8 -*-
#
# Copyright (C) 2023 Benjamin Thomas Schwertfeger
# All rights reserved.
# https://github.com/btschwertfeger
#

"""
Module that provides an example usage for the KrakenSpotWebsocketClient.
It uses the Kraken Websocket API v2.
"""

from __future__ import annotations

```

(continues on next page)

(continued from previous page)

```python
import asyncio
import logging
import os

from kraken.spot import SpotWSClient

logging.basicConfig(
    format="%(asctime)s %(module)s,line: %(lineno)d %(levelname)8s | %(message)s",
    datefmt="%Y/%m/%d %H:%M:%S",
    level=logging.INFO,
)
logging.getLogger("requests").setLevel(logging.WARNING)
logging.getLogger("urllib3").setLevel(logging.WARNING)

clients = []


class Client(SpotWSClient):
    """Can be used to create a custom trading strategy"""

    async def on_message(self: Client, message: dict) -> None:
        """Receives the websocket messages"""
        if message.get("method") == "pong" or message.get("channel") == "heartbeat":
            return

        print(message)
        # now you can access lots of methods, for example to create an order:
        # if self._is_auth:  # only if the client is authenticated ...
        #     await self.send_message(
        #         message={
        #             "method": "add_order",
        #             "params": {
        #                 "limit_price": 1234.56,
        #                 "order_type": "limit",
        #                 "order_userref": 123456789,
        #                 "order_qty": 1.0,
        #                 "side": "buy",
        #                 "symbol": "BTC/USD",
        #                 "validate": True,
        #             },
        #         }
        #     )
        # ... it is also possible to call regular REST endpoints
        # but using the websocket messages is more efficient.
        # You can also un-/subscribe here using self.subscribe/self.unsubscribe.


async def main() -> None:
    key: str = os.getenv("SPOT_API_KEY")
    secret: str = os.getenv("SPOT_SECRET_KEY")

    try:
```

(continues on next page)

(continued from previous page)

```
        # Public/unauthenticated websocket client
        client: Client = Client()  # only use this one if you don't need private feeds
        clients.append(client)
        await client.start()
        # print(client.public_channel_names)  # list public subscription names

        await client.subscribe(
            params={"channel": "ticker", "symbol": ["BTC/USD", "DOT/USD"]},
        )
        await client.subscribe(
            params={"channel": "book", "depth": 25, "symbol": ["BTC/USD"]},
        )
        # await client.subscribe(params={"channel": "ohlc", "symbol": ["BTC/USD"]})
        await client.subscribe(
            params={
                "channel": "ohlc",
                "interval": 15,
                "snapshot": False,
                "symbol": ["BTC/USD", "DOT/USD"],
            },
        )
        await client.subscribe(params={"channel": "trade", "symbol": ["BTC/USD"]})

        # wait because unsubscribing is faster than unsubscribing ... (just for that␣
↪example)
        await asyncio.sleep(3)
        # print(client.active_public_subscriptions) # ... to list active subscriptions
        await client.unsubscribe(
            params={"channel": "ticker", "symbol": ["BTC/USD", "DOT/USD"]},
        )
        # ...

        if key and secret:
            # Per default, the authenticated client starts two websocket connections,
            # one for authenticated and one for public messages. If there is no need
            # for a public connection, it can be disabled using the ``no_public``
            # parameter.
            client_auth = Client(key=key, secret=secret, no_public=True)
            clients.append(client_auth)
            await client_auth.start()
            # print(client_auth.private_channel_names)  # ... list private channel names
            # when using the authenticated client, you can also subscribe to public feeds
            await client_auth.subscribe(params={"channel": "executions"})

            await asyncio.sleep(5)
            await client_auth.unsubscribe(params={"channel": "executions"})

        while not client.exception_occur:  # and not client_auth.exception_occur:
            await asyncio.sleep(6)
    finally:
        # Stop the sessions properly.
        for open_client in clients:
```

(continues on next page)

(continued from previous page)

```
            await open_client.close()


if __name__ == "__main__":
    asyncio.run(main())

# ============================================================
# Alternative - as ContextManager:

# from kraken.spot import SpotWSClient
# import asyncio


# async def on_message(message: dict) -> None:
#     print(message)


# async def main() -> None:
#     async with SpotWSClient(callback=on_message) as session:
#         await session.subscribe(params={"channel": "ticker", "symbol": ["BTC/USD"]})

#     while True:
#         await asyncio.sleep(6)


# if __name__ == "__main__":
#     try:
#         asyncio.run(main())
#     except KeyboardInterrupt:
#         pass
```

4.3 xStocks REST

The examples presented below serve to demonstrate the usage of the Spot REST clients provided by python-kraken-sdk to access Kraken's REST API for trading xStocks.

For questions, feedback, additions, suggestions for improvement or problems python-kraken-sdk/discussions or python-kraken-sdk/issues may be helpful.

See https://docs.kraken.com/api/docs/guides/global-intro for information about the available endpoints and their usage.

Listing 6: Example usage of Spot REST client for xStocks

```
# !/usr/bin/env python3
# -*- mode: python; coding: utf-8 -*-
#
# Copyright (C) 2025 Benjamin Thomas Schwertfeger
# All rights reserved.
# https://github.com/btschwertfeger
#

"""
```

(continues on next page)

(continued from previous page)

```
Module that implements *some* examples for the Kraken Spot REST clients usage
with focus on xStocks.

NOTE: The xStocks feature is not available globally. Please checkout Kraken's
      documentation to get to know the availability zones.
"""

import logging
import os

from kraken.spot import SpotClient, Trade

logging.basicConfig(
    format="%(asctime)s %(module)s,line: %(lineno)d %(levelname)8s | %(message)s",
    datefmt="%Y/%m/%d %H:%M:%S",
    level=logging.INFO,
)
logging.getLogger("requests").setLevel(logging.WARNING)
logging.getLogger("urllib3").setLevel(logging.WARNING)


key = os.getenv("SPOT_API_KEY")
secret = os.getenv("SPOT_SECRET_KEY")

CLIENT = SpotClient(key=key, secret=secret)


def list_all_xstocks() -> None:
    """List all available xStocks on Kraken."""
    print(
        CLIENT.request(
            "GET",
            "/0/public/AssetPairs",
            params={"aclass_base": "tokenized_asset"},  # <- important!
            auth=False,
        ),
    )


def create_xstock_order() -> None:
    """Create a test order for an xStock (validate mode, not placing actual order)."""
    print(
        CLIENT.request(
            "POST",
            "/0/private/AddOrder",
            params={
                "type": "buy",
                "volume": "1",
                "ordertype": "limit",
                "pair": "AAPLxUSD",
                "price": "100.0",
                "validate": True,
```

(continues on next page)

(continued from previous page)

```
                "asset_class": "tokenized_asset",  # <- important
            },
        ),
    )


def create_xstock_order_alternative() -> None:
    """Create a test order for an xStock (validate mode, not placing actual order)."""
    trade = Trade(key=key, secret=secret)

    print(
        trade.create_order(
            pair="AAPLxUSD",
            side="buy",
            ordertype="limit",
            volume="1",
            price="100.0",
            validate=True,
            extra_params={"asset_class": "tokenized_asset"},
        ),
    )


def main() -> None:
    """Uncomment the examples you want to run:"""
    # NOTE: These are only examples that show how to use the clients, there are
    #       many other functions available in the clients.
    list_all_xstocks()
    # create_xstock_order()
    # create_xstock_order_alternative()


if __name__ == "__main__":
    main()
```

4.4 Plotting Market Data

```
pip install matplotlib numpy pands ipykernel
python3 -m ipykernel install --user --name=kraken
```

[1]:
```
from kraken.spot import Market
```

[2]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

4.4.1 1. Create the unauthenticated market client

[3]:
```
market = Market()
```

4.4.2 2. Get the OHLC data of the XBTUSD pair, create the dataframe, and set the time index

[4]:
```
df = pd.DataFrame(
    market.get_ohlc(pair="XBTUSD", interval=60)["XXBTZUSD"],
    columns=["time", "open", "high", "low", "close", "vwap", "volume", "count"],
).astype(float)
# df = df.set_index('time')
df["time"] = pd.to_datetime(df["time"], unit="s")
df = df.sort_values(by="time")
df
```

[4]:
```
                  time     open     high      low    close     vwap  \
0   2024-03-02 12:00:00  61917.8  62223.7  61911.4  62142.4  62048.6
1   2024-03-02 13:00:00  62142.4  62142.4  61900.0  61900.0  62006.8
2   2024-03-02 14:00:00  61900.1  61971.8  61663.3  61891.7  61832.5
3   2024-03-02 15:00:00  61891.8  62032.2  61810.0  61836.8  61941.1
4   2024-03-02 16:00:00  61836.8  62022.3  61714.9  61930.8  61839.0
..                  ...      ...      ...      ...      ...      ...
715 2024-04-01 07:00:00  69680.1  69788.8  69562.0  69671.7  69676.0
716 2024-04-01 08:00:00  69671.8  69671.8  69244.7  69450.0  69445.2
717 2024-04-01 09:00:00  69450.1  69613.4  69448.6  69448.7  69510.3
718 2024-04-01 10:00:00  69448.7  69604.4  69353.0  69601.1  69454.0
719 2024-04-01 11:00:00  69601.2  69850.1  69601.1  69666.1  69732.8

         volume   count
0     15.260659   616.0
1     50.357137   727.0
2    281.512303  1538.0
3     38.815926   886.0
4    181.980102  1261.0
..          ...     ...
715   40.187037   767.0
716   91.352301  1017.0
717   46.275460   731.0
718   23.874609   556.0
719   20.962756   455.0

[720 rows x 8 columns]
```

4.4.3 3. Compute some indicatoes based on the loaded data

[5]:
```
# compute ema
df["ema21"] = df["close"].ewm(span=21, adjust=False, min_periods=21).mean()
df["ema50"] = df["close"].ewm(span=50, adjust=False, min_periods=50).mean()
df["ema200"] = df["close"].ewm(span=200, adjust=False, min_periods=200).mean()
```

[6]:
```
def support_and_resistance(df: pd.DataFrame, lookback: int = 200, levels: int = 3) ->␣
↪dict:
    """Returns up to 3 support and resistance levels by given dataframe"""
    high = df["high"][-lookback:].max()
    low = df["low"][-lookback:].min()
    close = df["close"][-lookback:].iloc[-1]

    pp = (high + low + close) / 3
    s1 = 2 * pp - high
    r1 = 2 * pp - low
    if levels >= 2:
        s2 = pp - (high - low)
        r2 = pp + (high - low)
        if levels >= 3:
            s3 = low - 2 * (high - pp)
            r3 = high + 2 * (pp - low)
            return {"s1": s1, "s2": s2, "s3": s3, "r1": r1, "r2": r2, "r3": r3}
        return {"s1": s1, "s2": s2, "r1": r1, "r2": r2}
    return {"s1": s1, "r2": r1}
```

[7]:
```
# compute support and resistance levels
for i, row in enumerate(df.index):
    try:
        srlevels = support_and_resistance(df.iloc[:i], lookback=50, levels=2)
        df.at[row, "s1"] = srlevels["s1"]
        df.at[row, "r1"] = srlevels["r1"]
        df.at[row, "s2"] = srlevels["s2"]
        df.at[row, "r2"] = srlevels["r2"]
    except:  # noqa: E722
        df.at[row, "s1"] = np.NaN
        df.at[row, "r1"] = np.NaN
        df.at[row, "s2"] = np.NaN
        df.at[row, "r2"] = np.NaN

# isn't that a beauty?:
df
```

[7]:

	time	open	high	low	close	vwap	volume	count	ema21	ema50	ema200
0	2024-03-02 12:00:00	61917.8	62223.7	61911.4	62142.4	62048.6	15.260659	616.0	NaN	NaN	NaN
1	2024-03-02 13:00:00	62142.4	62142.4	61900.0	61900.0	62006.8	50.357137	727.0	NaN	NaN	NaN
2	2024-03-02 14:00:00	61900.1	61971.8	61663.3	61891.7	61832.5					
3	2024-03-02 15:00:00	61891.8	62032.2	61810.0	61836.8	61941.1					
4	2024-03-02 16:00:00	61836.8	62022.3	61714.9	61930.8	61839.0					
..	...	...	...	...	...	...					
715	2024-04-01 07:00:00	69680.1	69788.8	69562.0	69671.7	69676.0					
716	2024-04-01 08:00:00	69671.8	69671.8	69244.7	69450.0	69445.2					
717	2024-04-01 09:00:00	69450.1	69613.4	69448.6	69448.7	69510.3					
718	2024-04-01 10:00:00	69448.7	69604.4	69353.0	69601.1	69454.0					
719	2024-04-01 11:00:00	69601.2	69850.1	69601.1	69666.1	69732.8					

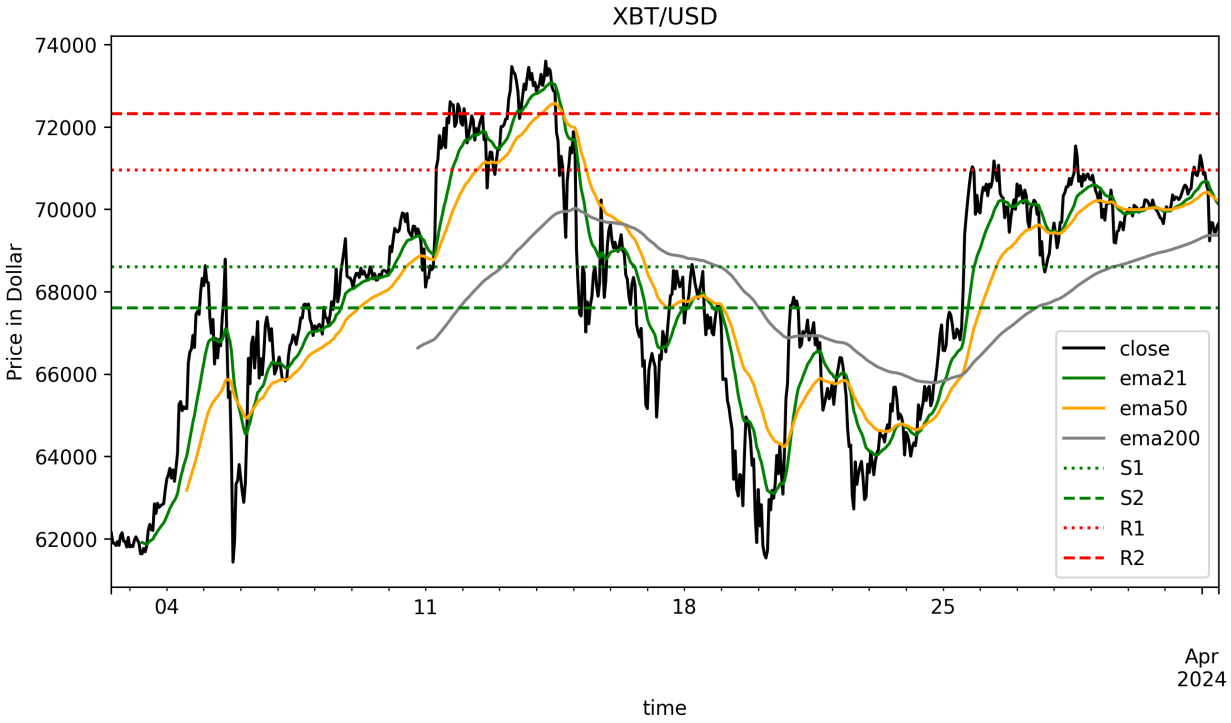
(continues on next page)

(continued from previous page)

```
2    281.512303  1538.0           NaN           NaN           NaN
3     38.815926   886.0           NaN           NaN           NaN
4    181.980102  1261.0           NaN           NaN           NaN
..          ...     ...           ...           ...           ...
715   40.187037   767.0  70380.082934  70315.531093  69369.034227
716   91.352301  1017.0  70295.529940  70281.588697  69369.839856
717   46.275460   731.0  70218.545400  70248.926395  69370.624534
718   23.874609   556.0  70162.414000  70223.521439  69372.917823
719   20.962756   455.0  70117.294545  70201.661774  69375.835058

               s1            r1            s2            r2
0             NaN           NaN           NaN           NaN
1    61961.300000  62273.600000  61780.200000  62404.800000
2    61792.100000  62115.800000  61684.200000  62331.600000
3    61628.766667  62189.166667  61365.833333  62486.633333
4    61592.166667  62152.566667  61347.533333  62468.333333
..            ...           ...           ...           ...
715  68659.500000  71012.200000  67639.000000  72344.400000
716  68653.966667  71006.666667  67636.233333  72341.633333
717  68506.166667  70858.866667  67562.333333  72267.733333
718  68505.300000  70858.000000  67561.900000  72267.300000
719  68606.900000  70959.600000  67612.700000  72318.100000

[720 rows x 15 columns]
```

4.4.4 4. Plot the results

[8]:
```
fig = plt.figure(figsize=(10, 5), dpi=300)
ax = plt.gca()
df.plot(x="time", y=["close", "ema21", "ema50", "ema200"], ax=ax, color=["black", "green
↪", "orange", "gray"])

xmin, xmax = df["time"].iloc[0], df["time"].iloc[-1]
ax.hlines(y=df["s1"].iloc[-1], xmin=xmin, xmax=xmax, color="green", linestyle=":", label=
↪"S1")
ax.hlines(y=df["s2"].iloc[-1], xmin=xmin, xmax=xmax, color="green", linestyle="--",␣
↪label="S2")
ax.hlines(y=df["r1"].iloc[-1], xmin=xmin, xmax=xmax, color="red", linestyle=":", label=
↪"R1")
ax.hlines(y=df["r2"].iloc[-1], xmin=xmin, xmax=xmax, color="red", linestyle="--", label=
↪"R2")
ax.set_ylabel("Price in Dollar")
plt.legend()
plt.title("XBT/USD");
```



… Create and combine custom indicators, implement them, and build your own strategies …

[ ]:

4.5 Futures Websocket

The examples presented below serve to demonstrate the usage of the Futures websocket clients provided by python-kraken-sdk to access Kraken's Websocket API.

For questions, feedback, additions, suggestions for improvement or problems python-kraken-sdk/discussions or python-kraken-sdk/issues may be helpful.

Listing 7: Example access and usage for Kraken Futures Websocket API

```
# !/usr/bin/env python3
# -*- mode: python; coding: utf-8 -*-
#
# Copyright (C) 2023 Benjamin Thomas Schwertfeger
# All rights reserved.
# https://github.com/btschwertfeger
#

"""
Module that provides an example usage for the Kraken Futures websocket client.
"""

from __future__ import annotations

```

(continues on next page)

(continued from previous page)

```python
import asyncio
import logging
import os
import time

from kraken.futures import FuturesWSClient

logging.basicConfig(
    format="%(asctime)s %(module)s,line: %(lineno)d %(levelname)8s | %(message)s",
    datefmt="%Y/%m/%d %H:%M:%S",
    level=logging.INFO,
)
logging.getLogger("requests").setLevel(logging.WARNING)
logging.getLogger("urllib3").setLevel(logging.WARNING)
LOG: logging.Logger = logging.getLogger(__name__)

clients = []


# Custom client
class Client(FuturesWSClient):
    """Can be used to create a custom trading strategy"""

    async def on_message(self: Client, message: list | dict) -> None:
        """Receives the websocket messages"""
        LOG.info(message)
        # ... apply your trading strategy in this class
        # ... you can also combine this with the Futures REST clients


async def main() -> None:
    """Create a client and subscribe to channels/feeds"""

    key = os.getenv("FUTURES_API_KEY")
    secret = os.getenv("FUTURES_SECRET_KEY")

    try:
        # _____Public_Websocket_Feeds___________________
        client = Client()
        clients.append(client)
        await client.start()
        # print(client.get_available_public_subscription_feeds())

        products = ["PI_XBTUSD", "PF_SOLUSD"]
        # subscribe to a public websocket feed
        await client.subscribe(feed="ticker", products=products)
        await client.subscribe(feed="book", products=products)
        # await client.subscribe(feed='trade', products=products)
        # await client.subscribe(feed='ticker_lite', products=products)
        # await client.subscribe(feed='heartbeat')
        # time.sleep(2)

```

(continues on next page)

(continued from previous page)

```
        # unsubscribe from a websocket feed
        time.sleep(2)  # in case subscribe is not done yet
        # await client.unsubscribe(feed='ticker', products=['PI_XBTUSD'])
        await client.unsubscribe(feed="ticker", products=["PF_XBTUSD"])
        await client.unsubscribe(feed="book", products=products)
        # ...

        # _____Private_Websocket_Feeds_________________
        if key and secret:
            client_auth = Client(key=key, secret=secret)
            clients.append(client_auth)
            await client_auth.start()
            # print(client_auth.get_available_private_subscription_feeds())

            # subscribe to a private/authenticated websocket feed
            await client_auth.subscribe(feed="fills")
            await client_auth.subscribe(feed="open_positions")
            # await client_auth.subscribe(feed='open_orders')
            # await client_auth.subscribe(feed='open_orders_verbose')
            # await client_auth.subscribe(feed='deposits_withdrawals')
            # await client_auth.subscribe(feed='account_balances_and_margins')
            # await client_auth.subscribe(feed='balances')
            # await client_auth.subscribe(feed='account_log')
            # await client_auth.subscribe(feed='notifications_auth')

            # authenticated clients can also subscribe to public feeds
            # await client_auth.subscribe(feed='ticker', products=['PI_XBTUSD', 'PF_ETHUSD'])

            # time.sleep(1)
            # unsubscribe from a private/authenticated websocket feed
            await client_auth.unsubscribe(feed="fills")
            await client_auth.unsubscribe(feed="open_positions")
            # ...

        while not client.exception_occur:  # and not client_auth.exception_occur:
            await asyncio.sleep(6)
    finally:
        # Close the sessions properly.
        for open_client in clients:
            await open_client.close()


if __name__ == "__main__":
    asyncio.run(main())
    # the websocket client will send {'event': 'asyncio.CancelledError'} via on_message
    # so you can handle the behavior/next actions individually within you strategy

# ====================================================================
# Alternative - as ContextManager:

# from kraken.futures import KrakenFuturesWSClient
# import asyncio
```

(continues on next page)

(continued from previous page)

```
# async def on_message(message):
#     print(message)

# async def main() -> None:
#     async with KrakenFuturesWSClient(callback=on_message) as session:
#         await session.subscribe(feed="ticker", products=["PF_XBTUSD"])
#     while True:
#         await asyncio.sleep(6)

# if __name__ == "__main__":
#     try:
#         asyncio.run(main())
#     except KeyboardInterrupt:
#         pass
```

4.6 Futures REST

The examples presented below serve to demonstrate the usage of the Futures REST clients provided by python-kraken-sdk to access Kraken's REST API.

For questions, feedback, additions, suggestions for improvement or problems python-kraken-sdk/discussions or python-kraken-sdk/issues may be helpful.

See https://docs.kraken.com/api/docs/guides/global-intro for information about the available endpoints and their usage.

The Futures client provides access to all un-and authenticated endpoints of Kraken's Futures API.

Listing 8: Example: Spot Client Usage (1)

```
from kraken.futures import FuturesClient

client = FuturesClient(key="<your-api-key>", secret="<your-secret-key>")
print(client.request("GET", "/derivatives/api/v3/accounts"))
```

The async Futures client allows for asynchronous access to Kraken's Futures endpoints. Below are two examples demonstrating its usage.

Using FuturesAsyncClient without a context manager; In this example, the client is manually closed after the request is made.

Listing 9: Example: Spot Client Usage (2)

```
import asyncio
from kraken.futures import FuturesAsyncClient

async def main():
    client = FuturesAsyncClient(key="<your-api-key>", secret="<your-secret-key>")
    try:
        response = await client.request("GET", "/derivatives/api/v3/accounts")
        print(response)
    finally:
        await client.async_close()
```

(continues on next page)

(continued from previous page)

```
11
12 asyncio.run(main())
```

Using FuturesAsyncClient as context manager; This example demonstrates the use of the context manager, which ensures the client is automatically closed after the request is completed.

Listing 10: Example: Spot Client Usage (3)

```
1  import asyncio
2  from kraken.futures import FuturesAsyncClient
3
4  async def main():
5     async with FuturesAsyncClient(
6        key="<your-api-key>", secret="<your-secret-key>"
7     ) as client:
8        response = await client.request("GET", "/derivatives/api/v3/accounts")
9        print(response)
10
11 asyncio.run(main())
```

The following legacy examples are not maintained on a regular basis. They serve only for demonstration purposes - make sure to checkout the documentation of the individual functions.

Listing 11: Example usage of Futures REST clients

```
1  # !/usr/bin/env python3
2  # -*- mode: python; coding: utf-8 -*-
3  #
4  # Copyright (C) 2023 Benjamin Thomas Schwertfeger
5  # All rights reserved.
6  # https://github.com/btschwertfeger
7  #
8
9  """
10 Module that implements *some* examples for the Kraken Futures REST clients
11 usage.
12 """
13
14 import logging
15 import os
16 import time
17 from pathlib import Path
18
19 from kraken.futures import Funding, Market, Trade, User
20
21 logging.basicConfig(
22     format="%(asctime)s %(module)s,line: %(lineno)d %(levelname)8s | %(message)s",
23     datefmt="%Y/%m/%d %H:%M:%S",
24     level=logging.INFO,
25 )
26 logging.getLogger("requests").setLevel(logging.WARNING)
27 logging.getLogger("urllib3").setLevel(logging.WARNING)
28
```

(continues on next page)

(continued from previous page)

```
key = os.getenv("FUTURES_SANDBOX_KEY")
secret = os.getenv("FUTURES_SANDBOX_SECRET")


def market_examples() -> None:
    """Example Futures Market client usage"""

    # Usage of the Market client to access public endpoints:
    market = Market()
    print(market.get_tick_types())
    print(market.get_tradeable_products(tick_type="trade"))
    print(market.get_resolutions(tick_type="trade", tradeable="PI_XBTUSD"))
    print(
        market.get_ohlc(
            tick_type="trade",
            symbol="PI_XBTUSD",
            resolution="5m",
            from_="1668989233",
        ),
    )
    print(market.get_fee_schedules())
    print(
        market.get_orderbook(symbol="fi_xbtusd_180615"),
    )  # might need adjustment of the symbol
    print(market.get_tickers())
    print(market.get_instruments())
    print(market.get_instruments_status())
    print(market.get_instruments_status(instrument="PI_XBTUSD"))
    print(market.get_trade_history(symbol="PI_XBTUSD"))
    print(market.get_historical_funding_rates(symbol="PI_XBTUSD"))
    time.sleep(2)  # Just to avoid rate limits

    # Usage of the Market client to access private endpoints:
    # (commented out to avoid accidental usage)
    priv_market = Market(key=key, secret=secret, sandbox=True)
    # print(priv_market.get_fee_schedules_vol())
    print(priv_market.get_leverage_preference())
    # print(priv_market.set_leverage_preference(symbol='PF_XBTUSD', maxLeverage=2)) # set␣
↪max leverage
    # print(priv_market.set_leverage_preference(symbol='PF_XBTUSD')) # reset max leverage
    # print(priv_market.set_pnl_preference(symbol='PF_XBTUSD', pnlPreference='BTC'))

    # time.sleep(2)
    # print(priv_market.get_execution_events())
    # print(market.get_public_execution_events(tradeable='PI_XBTUSD'))
    # print(market.get_public_order_events(tradeable='PI_XBTUSD'))
    # print(market.get_public_mark_price_events(tradeable='PI_XBTUSD'))
    # print(priv_market.get_order_events())
    # print(priv_market.get_trigger_events())


def user_examples() -> None:
```

(continues on next page)

(continued from previous page)

```
    """Example Futures User client usage"""
    # NOTE: This only works if you have set valid credentials for the the
    #       Futures demo environment. Remove the `sandbox=True` argument to use
    #       the production environment.
    #
    # Usage of the User client to access private endpoints:
    user = User(key=key, secret=secret, sandbox=True)
    print(user.get_wallets())
    print(user.get_subaccounts())
    print(user.get_unwind_queue())
    print(user.get_notifications())
    print(user.get_open_positions())
    print(user.get_open_orders())

    # You can retrieve the account log like so:
    print(user.get_account_log(before="1604937694000"))
    print(user.get_account_log(info="futures liquidation"))
    time.sleep(2)  # Just to avoid rate limits

    response = user.get_account_log_csv()
    assert response.status_code in {200, "200"}
    with Path("account_log.csv").open("wb") as file:
        for chunk in response.iter_content(chunk_size=512):
            if chunk:
                file.write(chunk)


def trade_examples() -> None:
    """Example Futures Trade client usage"""
    print(
        "Attention: Please check if you want to execute the trade endpoints!"
        " Check the script manually before running this example.",
    )
    return
    # return
    # NOTE: This only works if you have set valid credentials for the the
    #       Futures demo environment. Remove the `sandbox=True` argument to use
    #       the production environment.
    trade = Trade(key=key, secret=secret, sandbox=True)
    print(trade.get_fills())
    print(trade.get_fills(lastFillTime="2020-07-21T12:41:52.790Z"))
    print(
        trade.create_batch_order(
            batchorder_list=[
                {
                    "order": "send",
                    "order_tag": "1",
                    "orderType": "lmt",
                    "symbol": "PI_XBTUSD",
                    "side": "buy",
                    "size": 1,
                    "limitPrice": 1.00,
```

(continues on next page)

(continued from previous page)

```
                },
                {
                    "order": "send",
                    "order_tag": "2",
                    "orderType": "stp",
                    "symbol": "PI_XBTUSD",
                    "side": "buy",
                    "size": 1,
                    "limitPrice": 2.00,
                    "stopPrice": 3.00,
                },
                {
                    "order": "cancel",
                    "order_id": "e35d61dd-8a30-4d5f-a574-b5593ef0c050",
                },
                {
                    "order": "cancel",
                    "cliOrdId": 123456789,
                },
            ],
        ),
    )
    print(trade.cancel_all_orders())
    print(trade.cancel_all_orders(symbol="pi_xbtusd"))
    print(trade.dead_mans_switch(timeout=60))
    print(trade.dead_mans_switch(timeout=0))  # to deactivate
    print(trade.cancel_order(order_id="some order id"))
    print(
        trade.edit_order(
            orderId="some order id",
            size=300,
            limitPrice=401,
            stopPrice=350,
        ),
    )
    print(trade.get_orders_status(orderIds=["orderid1", "orderid2"]))
    print(
        trade.create_order(
            orderType="lmt",
            side="buy",
            size=1,
            limitPrice=4,
            symbol="pf_bchusd",
        ),
    )
    print(
        trade.create_order(
            orderType="take_profit",
            side="buy",
            size=1,
            symbol="pf_bchusd",
            stopPrice=100,
```

(continues on next page)

(continued from previous page)

```
            triggerSignal="mark",
        ),
    )


def funding_examples() -> None:
    """Example Funding client usage"""
    funding = Funding(key=key, secret=secret, sandbox=True)
    print(funding.get_historical_funding_rates(symbol="PF_SOLUSD"))


def main() -> None:
    """Uncomment the examples you want to run:"""
    # user_examples()
    # market_examples()
    # trade_examples()
    # funding_examples()


if __name__ == "__main__":
    main()
```

CHAPTER

FIVE

TRADING BOT TEMPLATES

The python-kraken-sdk is perfectly suited to develop automated trading algorithms. To facilitate the start here directly for Spot trading as well as for the Futures enthusiasts, templates are provided which can serve as the basis for dynamic trading algorithms. In both cases websockets are used to capture the live data. In addition REST clients are integrated, so that their endpoints can also be reached at any time.

For questions, feedback, additions, suggestions or problems python-kraken-sdk/discussions or python-kraken-sdk/issues may be helpful.

5.1 Spot Trading Bot Templates

The templates presented below serve as starting points for the development of a trading algorithms for Spot trading on the crypto asset exchange Kraken using the python-kraken-sdk.

The trading strategy can be implemented using the `TradingBot` class. This class has access to all REST clients and receives all messages that are sent by the subscribed websocket feeds via the `on_message` function.

Listing 1: Template to build a trading bot using the Kraken Spot Websocket API v2

```
# !/usr/bin/env python3
# -*- mode: python; coding: utf-8 -*-
#
# Copyright (C) 2023 Benjamin Thomas Schwertfeger
# All rights reserved.
# https://github.com/btschwertfeger
#

"""
Module that provides a template to build a Spot trading algorithm using the
python-kraken-sdk and Kraken Spot websocket API v2.
"""

from __future__ import annotations

import asyncio
import logging
import os
import sys
import traceback

import requests
```

(continues on next page)

(continued from previous page)

```python
import urllib3

from kraken.exceptions import KrakenAuthenticationError  # , KrakenPermissionDeniedError
from kraken.spot import Funding, Market, SpotWSClient, Trade, User

logging.basicConfig(
    format="%(asctime)s %(module)s,line: %(lineno)d %(levelname)8s | %(message)s",
    datefmt="%Y/%m/%d %H:%M:%S",
    level=logging.INFO,
)
logging.getLogger("requests").setLevel(logging.WARNING)
logging.getLogger("urllib3").setLevel(logging.WARNING)

LOG: logging.Logger = logging.getLogger(__name__)


class TradingBot(SpotWSClient):
    """
    Class that implements the trading strategy

    * The on_message function gets all messages sent by the websocket feeds.
    * Decisions can be made based on these messages
    * Can place trades using the self.__trade client or self.send_message
    * Do everything you want

    ====== P A R A M E T E R S ======
    config: dict
        configuration like: {
            "key": "kraken-spot-key",
            "secret": "kraken-spot-secret",
            "pairs": ["DOT/USD", "BTC/USD"],
        }
    """

    def __init__(
        self: TradingBot,
        config: dict,
        **kwargs: object | dict | set | tuple | list | str | float | None,
    ) -> None:
        super().__init__(
            key=config["key"],
            secret=config["secret"],
            **kwargs,
        )
        self.__config: dict = config

        self.__user: User = User(key=config["key"], secret=config["secret"])
        self.__trade: Trade = Trade(key=config["key"], secret=config["secret"])
        self.__market: Market = Market(key=config["key"], secret=config["secret"])
        self.__funding: Funding = Funding(key=config["key"], secret=config["secret"])

    async def on_message(self: TradingBot, message: dict) -> None:
```

(continues on next page)

(continued from previous page)

```
        """Receives all messages of the websocket connection(s)"""
        if message.get("method") == "pong" or message.get("channel") == "heartbeat":
            return
        if "error" in message:
            # handle exceptions/errors sent by websocket connection ...
            pass

        LOG.info(message)

        # == apply your trading strategy here ==

        # Call functions of `self.__trade` and other clients if conditions met ...
        # try:
        #     print(self.__trade.create_order(
        #         ordertype='limit',
        #         side='buy',
        #         volume=2,
        #         pair='XBTUSD',
        #         price=12000
        #     ))
        # except KrakenPermissionDeniedError:
        #    # ... handle exceptions
        #    pass

        # The spot websocket client also allow sending orders via websockets
        # this is way faster than using REST endpoints.
        # await self.send_message(
        #     message={
        #         "method": "add_order",
        #         "params": {
        #             "limit_price": 1234.56,
        #             "order_type": "limit",
        #             "order_userref": 123456789,
        #             "order_qty": 1.0,
        #             "side": "buy",
        #             "symbol": "BTC/USD",
        #             "validate": True,
        #         },
        #     }
        # )

        # You can also un-/subscribe here using `self.subscribe(...)` or
        # `self.unsubscribe(...)`.
        #
        # ... more can be found in the documentation
        #        (https://python-kraken-sdk.readthedocs.io/en/stable/).

    # Add more functions to customize the trading strategy ...

    def save_exit(self: TradingBot, reason: str | None = "") -> None:
        """controlled shutdown of the strategy"""
        LOG.warning("Save exit triggered, reason: %s", reason)
```

(continues on next page)

(continued from previous page)

```
        # Some ideas:
        #   * Save the current data
        #   * Close trades
        #   * Enable dead man's switch
        sys.exit(1)


class Manager:
    """
    Class to manage the trading strategy

    ... subscribes to desired feeds, instantiates the strategy and runs as long
    as there is no error.

    ====== P A R A M E T E R S ======
    config: dict
        configuration like: {
            "key" "kraken-spot-key",
            "secret": "kraken-secret-key",
            "pairs": ["DOT/USD", "BTC/USD"],
        }
    """

    def __init__(self: Manager, config: dict) -> None:
        self.__config: dict = config
        self.__trading_strategy: TradingBot | None = None

    def run(self: Manager) -> None:
        """Starts the event loop and bot"""
        if not self.__check_credentials():
            sys.exit(1)

        try:
            asyncio.run(self.__main())
        except KeyboardInterrupt:
            self.save_exit(reason="KeyboardInterrupt")
        else:
            self.save_exit(reason="Asyncio loop left")

    async def __main(self: Manager) -> None:
        """
        Instantiates the trading strategy and subscribes to the desired
        websocket feeds. While no exception within the strategy occur run the
        loop.

        The variable `exception_occur` which is an attribute of the SpotWSClient
        can be set individually but is also being set to `True` if the websocket
        connection has some fatal error. This is used to exit the asyncio loop -
        but you can also apply your own reconnect rules.
        """
        try:
            self.__trading_strategy = TradingBot(config=self.__config)
```

(continues on next page)

(continued from previous page)

```
            await self.__trading_strategy.start()

            await self.__trading_strategy.subscribe(
                params={"channel": "ticker", "symbol": self.__config["pairs"]},
            )
            await self.__trading_strategy.subscribe(
                params={
                    "channel": "ohlc",
                    "interval": 15,
                    "symbol": self.__config["pairs"],
                },
            )

            await self.__trading_strategy.subscribe(params={"channel": "executions"})

            while not self.__trading_strategy.exception_occur:
                # Check if the algorithm feels good
                # Send a status update every day via Telegram or Mail
                # ...
                await asyncio.sleep(6)

        except Exception as exc:
            LOG.error(message := f"Exception in main: {exc} {traceback.format_exc()}")
            self.__trading_strategy.save_exit(reason=message)
        finally:
            # Close the sessions properly.
            await self.__trading_strategy.close()

    def __check_credentials(self: Manager) -> bool:
        """Checks the user credentials and the connection to Kraken"""
        try:
            User(self.__config["key"], self.__config["secret"]).get_account_balance()
            LOG.info("Client credentials are valid.")
            return True
        except urllib3.exceptions.MaxRetryError:
            LOG.error("MaxRetryError, can't connect.")
            return False
        except requests.exceptions.ConnectionError:
            LOG.error("ConnectionError, Kraken not available.")
            return False
        except KrakenAuthenticationError:
            LOG.error("Invalid credentials!")
            return False

    def save_exit(self: Manager, reason: str = "") -> None:
        """Invoke the save exit function of the trading strategy"""
        print(f"Save exit triggered - {reason}")
        if self.__trading_strategy is not None:
            self.__trading_strategy.save_exit(reason=reason)
        else:
            sys.exit(1)

```

(continues on next page)

(continued from previous page)

```python

def main() -> None:
    """Example main - load environment variables and run the strategy."""
    manager: Manager = Manager(
        config={
            "key": os.getenv("SPOT_API_KEY"),
            "secret": os.getenv("SPOT_SECRET_KEY"),
            "pairs": ["DOT/USD", "BTC/USD"],
        },
    )

    try:
        manager.run()
    except Exception:
        manager.save_exit(
            reason=f"manageBot.run() has ended: {traceback.format_exc()}",
        )


if __name__ == "__main__":
    main()
```

5.2 Maintain a valid Spot Orderbook

The following examples demonstrate how to use the python-kraken-sdk to retrieve valid realtime orderbooks. The current implementation of the *kraken.spot.SpotOrderBookClient* uses the websocket API v2.

Listing 2: Sample on how to maintain a valid orderbook w/ websocket API

```python
# !/usr/bin/env python3
# -*- mode: python; coding: utf-8 -*-
#
# Copyright (C) 2023 Benjamin Thomas Schwertfeger
# All rights reserved.
# https://github.com/btschwertfeger
#

"""

**For websocket API v2**

This module provides an example on how to use the Spot Orderbook client of the
python-kraken-sdk (https://github.com/btschwertfeger/python-kraken-sdk) to
retrieve and maintain a valid Spot order book for (a) specific asset pair(s).
It can be run directly without any credentials if the python-kraken-sdk is
installed.

    python3 -m pip install python-kraken-sdk

The output when running this snippet looks like the following table and updates
```

(continues on next page)

(continued from previous page)

```
the book as soon as Kraken sent any order book update.

Bid          Volume               Ask          Volume
27076.00000 (8.28552127)          27076.10000 (2.85897056)
27075.90000 (3.75748052)          27077.30000 (0.57243521)
27074.40000 (0.57249652)          27080.80000 (0.00100000)
27072.90000 (0.01200917)          27081.00000 (0.00012345)
27072.80000 (0.25000000)          27081.70000 (0.30000000)
27072.30000 (4.89735970)          27082.70000 (0.05539777)
27072.20000 (2.65896716)          27082.80000 (0.00400000)
27072.10000 (2.77037635)          27082.90000 (0.57231684)
27072.00000 (0.81770000)          27083.00000 (0.38934000)
27071.50000 (0.07194657)          27083.80000 (2.76918992)

This can be the basis of an order book based trading strategy where realtime
data and fast price movements are considered.
"""

from __future__ import annotations

import asyncio
import logging
from typing import Any

from kraken.spot import SpotOrderBookClient

logging.basicConfig(
    format="%(asctime)s %(module)s,line: %(lineno)d %(levelname)8s | %(message)s",
    datefmt="%Y/%m/%d %H:%M:%S",
    level=logging.INFO,
)
logging.getLogger().setLevel(logging.INFO)
logging.getLogger("requests").setLevel(logging.WARNING)


class Orderbook(SpotOrderBookClient):
    """
    This is a wrapper class that is used to overload the :func:`on_book_update`
    function. It can also be used as a base for trading strategy. Since the
    :class:`kraken.spot.SpotOrderBookClient` is derived from
    :class:`kraken.spot.SpotWSClient` it can also be used to access the
    :func:`subscribe` function and any other provided utility.
    """

    async def on_book_update(
        self: Orderbook,
        pair: str,
        message: list,  # noqa: ARG002
    ) -> None:
        """
        This function is called every time the order book of ``pair`` gets
        updated.
```

(continues on next page)

(continued from previous page)

```
        The ``pair`` parameter can be used to access the updated order book as
        shown in the function body below.

        :param pair: The currency pair of the updated order book
        :type pair: str
        :param message: The message sent by Kraken (not needed in most cases)
        :type message: list
        """
        book: dict[str, Any] = self.get(pair=pair)
        bid: list[tuple[str, str]] = list(book["bid"].items())
        ask: list[tuple[str, str]] = list(book["ask"].items())

        print("Bid     Volume\t\t  Ask     Volume")
        for level in range(self.depth):
            print(
                f"{bid[level][0]} ({bid[level][1][0]}) \t {ask[level][0]} (
↪{ask[level][1][0]})",
            )

        assert book["valid"]  # ensure that the checksum is valid
        # ... the client will automatically resubscribe to a book feed if the
        #   checksum is not valid. The user must not do anything for that, but
        #   will get informed.


async def main() -> None:
    """
    Here we depth of the order book and also a pair. We could
    subscribe to multiple pairs, but for simplicity only XBT/USD is chosen.

    The Orderbook class can be instantiated, which receives the order
    book-related messages, after we subscribed to the book feed.

    Finally we need some "game loop" - so we create a while loop
    that runs as long as there is no error.
    """

    async with Orderbook(depth=10) as orderbook:
        await orderbook.add_book(
            pairs=["BTC/USD"],  # we can also subscribe to more currency pairs
        )

        while not orderbook.exception_occur:
            await asyncio.sleep(10)


if __name__ == "__main__":
    try:
        asyncio.run(main())
    except KeyboardInterrupt:
        print("KeyboardInterrupt!")
```

References: - https://gist.github.com/btschwertfeger/6eea0eeff193f7cd1b262cfce4f0eb51

5.3 Futures Trading Bot Template

The template presented below serves as a starting point for the development of a trading algorithm for trading futures contracts on the crypto asset exchange Kraken using the python-kraken-sdk.

The trading strategy can be implemented in the `TradingBot` class. This class has access to all REST clients and receives all messages that are sent via the subscribed websocket feeds via the `on_message` function.

Listing 3: Template to build a trading bot using the Kraken Futures Websocket API

```python
# !/usr/bin/env python3
# -*- mode: python; coding: utf-8 -*-
#
# Copyright (C) 2023 Benjamin Thomas Schwertfeger
# All rights reserved.
# https://github.com/btschwertfeger
#

"""
Module that provides a template to build a Futures trading algorithm using the
python-kraken-sdk.
"""

from __future__ import annotations

import asyncio
import logging
import os
import sys
import traceback

import requests
import urllib3

from kraken.exceptions import KrakenAuthenticationError
from kraken.futures import FuturesWSClient, User

logging.basicConfig(
    format="%(asctime)s %(module)s,line: %(lineno)d %(levelname)8s | %(message)s",
    datefmt="%Y/%m/%d %H:%M:%S",
    level=logging.INFO,
)
logging.getLogger("requests").setLevel(logging.WARNING)
logging.getLogger("urllib3").setLevel(logging.WARNING)
LOG: logging.Logger = logging.getLogger(__name__)


class TradingBot(FuturesWSClient):
    """
    Class that implements the trading strategy
```

(continues on next page)

(continued from previous page)

```
    * The on_message function gets all messages sent by the websocket feeds.
    * Decisions can be made based on these messages
    * Can place trades using the self.__trade client
    * Do everything you want

    ====== P A R A M E T E R S ======
    config: dict
        configuration like: {
            'key' 'kraken-futures-key',
            'secret': 'kraken-secret-key',
            'products': ['PI_XBTUSD']
        }
    """

    def __init__(self: TradingBot, config: dict) -> None:
        super().__init__(  # initialize the KrakenFuturesWSClient
            key=config["key"],
            secret=config["secret"],
        )

    async def on_message(self: TradingBot, message: list | dict) -> None:
        """Receives all messages that came form the websocket feed(s)"""
        LOG.info(message)

        # == apply your trading strategy here ==
        # Hint: You can execute requests using the `request` function directly:
        # print(await self.request(
        #     "GET", "/api/charts/v1/spot/PI_XBTUSD/1d",
        # ))

        # You can also un-/subscribe here using `self.subscribe(...)` or
        # `self.unsubscribe(...)`
        # ... more can be found in the documentation
        #        (https://python-kraken-sdk.readthedocs.io/en/stable/).

    # Add more functions to customize the trading strategy ...

    def save_exit(self: TradingBot, reason: str = "") -> None:
        """Controlled shutdown of the strategy"""
        LOG.warning("Save exit triggered, reason: %s", reason)
        # Some ideas:
        #   * Save the bots data
        #   * Close trades
        #   * Enable dead man's switch
        sys.exit(1)


class ManagedBot:
    """
    Class to manage the trading strategy

```

(continues on next page)

(continued from previous page)

```
    ... subscribes to desired feeds, instantiates the strategy and runs as long
    as there is no error.

    ====== P A R A M E T E R S ======
    config: dict
        bot configuration like: {
            'key' 'kraken-futures-key',
            'secret': 'kraken-secret-key',
            'products': ['PI_XBTUSD']
        }
    """

    def __init__(self: ManagedBot, config: dict) -> None:
        self.__config: dict = config
        self.__trading_strategy: TradingBot | None = None

    def run(self: ManagedBot) -> None:
        """Runner function"""
        if not self.__check_credentials():
            sys.exit(1)

        try:
            asyncio.run(self.__main())
        except KeyboardInterrupt:
            self.save_exit(reason="KeyboardInterrupt")
        else:
            self.save_exit(reason="Asyncio loop left")

    async def __main(self: ManagedBot) -> None:
        """
        Instantiates the trading strategy/algorithm and subscribes to the
        desired websocket feeds. Run the loop while no exception occur.

        The variable `exception_occur` which is an attribute of the
        KrakenFuturesWSClient can be set individually but is also being set to
        `True` if the websocket connection has some fatal error. This is used to
        exit the asyncio loop - but you can also apply your own reconnect rules.
        """
        try:
            self.__trading_strategy = TradingBot(config=self.__config)
            await self.__trading_strategy.start()

            await self.__trading_strategy.subscribe(
                feed="ticker",
                products=self.__config["products"],
            )
            await self.__trading_strategy.subscribe(
                feed="book",
                products=self.__config["products"],
            )

            await self.__trading_strategy.subscribe(feed="fills")
```

(continues on next page)

(continued from previous page)

```
            await self.__trading_strategy.subscribe(feed="open_positions")
            await self.__trading_strategy.subscribe(feed="open_orders")
            await self.__trading_strategy.subscribe(feed="balances")

            while not self.__trading_strategy.exception_occur:
                # Check if the algorithm feels good
                # Send a status update every day via Telegram or Mail
                # ...
                await asyncio.sleep(6)

        except Exception as exc:
            LOG.error(message := f"Exception in main: {exc} {traceback.format_exc()}")
            self.__trading_strategy.save_exit(reason=message)
        finally:
            # Close the sessions properly.
            await self.__trading_strategy.close()

    def __check_credentials(self: ManagedBot) -> bool:
        """Checks the user credentials and the connection to Kraken"""
        try:
            User(self.__config["key"], self.__config["secret"]).get_wallets()
            LOG.info("Client credentials are valid.")
            return True
        except urllib3.exceptions.MaxRetryError:
            LOG.error("MaxRetryError, cannot connect.")
            return False
        except requests.exceptions.ConnectionError:
            LOG.error("ConnectionError, Kraken not available.")
            return False
        except KrakenAuthenticationError:
            LOG.error("Invalid credentials!")
            return False

    def save_exit(self: ManagedBot, reason: str = "") -> None:
        """Calls the save exit function of the trading strategy"""
        print(f"Save exit triggered - {reason}")
        if self.__trading_strategy is not None:
            self.__trading_strategy.save_exit(reason=reason)
        else:
            sys.exit(1)


def main() -> None:
    """Example main - load environment variables and run the strategy."""

    managed_bot: ManagedBot = ManagedBot(
        config={
            "key": os.getenv("FUTURES_API_KEY"),
            "secret": os.getenv("FUTURES_SECRET_KEY"),
            "products": ["PI_XBTUSD", "PF_SOLUSD"],
        },
    )
```

(continues on next page)

(continued from previous page)

```
    try:
        managed_bot.run()
    except Exception:
        managed_bot.save_exit(
            reason=f"manageBot.run() has ended: {traceback.format_exc()}",
        )


if __name__ == "__main__":
    main()
```

CHAPTER

SIX

SPOT REST CLIENTS

class kraken.base_api.**SpotClient**(*key: str = ''*, *secret: str = ''*, *url: str = ''*, *proxy: str | None = None*, ***, *use_custom_exceptions: bool = True*)

Bases: `object`

This class is the base for all Spot clients, handles un-/signed requests and returns exception handled results.

With this class you can easily interact with the Kraken Spot API, including trading Spot crypto assets, xStocks, and margin trading.

If you are facing timeout errors on derived clients, you can make use of the `TIMEOUT` attribute to deviate from the default `10` seconds.

Kraken sometimes rejects requests that are older than a certain time without further information. To avoid this, the session manager creates a new session every 5 minutes.

Parameters

- **key** (*str, optional*) – Spot API public key (default: "")
- **secret** (*str, optional*) – Spot API secret key (default: "")
- **url** (*str, optional*) – URL to access the Kraken API (default: https://api.kraken.com)
- **proxy** (*str, optional*) – proxy URL, may contain authentication information

get_nonce() → str

Return a new nonce

request(*method: str*, *uri: str*, *params: dict | None = None*, *timeout: int = 10*, ***, *auth: bool = True*, *do_json: bool = False*, *return_raw: bool = False*, *query_str: str | None = None*, *extra_params: str | dict | None = None*) → dict[str, Any] | list[str] | list[dict[str, Any]] | Response

Handles the requested requests, by sending the request, handling the response, and returning the message or in case of an error the respective Exception.

Parameters

- **method** (*str*) – The request method, e.g., `GET`, `POST`, `PUT`, ...
- **uri** (*str*) – The endpoint to send the message
- **auth** (*bool*) – If the requests needs authentication (default: `True`)
- **params** (*dict, optional*) – The query or post parameter of the request (default: `None`)
- **extra_params** (*str | dict, optional*) – Additional query or post parameter of the request (default: `None`)
- **timeout** (*int*) – Timeout for the request (default: `10`)
- **do_json** (*bool*) – If the `params` must be "jsonified" - in case of nested dict style

- **return_raw** (*bool, optional*) – If the response should be returned without parsing. This is used for example when requesting an export of the trade history as .zip archive.
- **query_str** (*str, optional*) – Add custom values to the query /0/public/Nfts?filter%5Bcollection_id%5D=NCQNABO-XYCA7-JMMSDF&page_size=10

Raises
kraken.exceptions.KrakenException.* – If the response contains errors

Returns
The response

Return type
dict | list | requests.Response

property return_unique_id: str

Returns a unique uuid string

Returns
uuid

Return type
str

class kraken.base_api.**SpotAsyncClient**(*key: str = ''*, *secret: str = ''*, *url: str = ''*, *proxy: str | None = None*, ***, *use_custom_exceptions: bool = True*)

Bases: *SpotClient*

This class provides the base client for accessing the Kraken Spot API using asynchronous requests.

If you are facing timeout errors on derived clients, you can make use of the `TIMEOUT` attribute to deviate from the default `10` seconds.

Kraken sometimes rejects requests that are older than a certain time without further information. To avoid this, the session manager creates a new session every 5 minutes.

Parameters

- **key** (*str, optional*) – Spot API public key (default: "")
- **secret** (*str, optional*) – Spot API secret key (default: "")
- **url** (*str, optional*) – URL to access the Kraken API (default: https://api.kraken.com)
- **proxy** (*str, optional*) – proxy URL, may contain authentication information

async_close() → None

Closes the aiohttp session

async close() → None

Closes the aiohttp session

get_nonce() → str

Return a new nonce

async request(*method: str*, *uri: str*, *params: dict | None = None*, *timeout: int = 10*, ***, *auth: bool = True*, *do_json: bool = False*, *return_raw: bool = False*, *query_str: str | None = None*, *extra_params: str | dict | None = None*) → Coroutine

Handles the requested requests, by sending the request, handling the response, and returning the message or in case of an error the respective Exception.

Parameters

- **method** (*str*) – The request method, e.g., GET, POST, PUT, ...
- **uri** (*str*) – The endpoint to send the message
- **auth** (*bool*) – If the requests needs authentication (default: True)
- **params** (*dict, optional*) – The query or post parameter of the request (default: None)
- **timeout** (*int*) – Timeout for the request (default: 10)
- **do_json** (*bool*) – If the params must be "jsonified" - in case of nested dict style
- **return_raw** (*bool, optional*) – If the response should be returned without parsing. This is used for example when requesting an export of the trade history as .zip archive.
- **query_str** (*str, optional*) – Add custom values to the query /0/public/Nfts?filter%5Bcollection_id%5D=NCQNABO-XYCA7-JMMSDF&page_size=10

Raises
kraken.exceptions.KrakenException.* – If the response contains errors

Returns
The response

Return type
dict | list | aiohttp.ClientResponse

property return_unique_id: str

Returns a unique uuid string

Returns
uuid

Return type
str

class kraken.spot.**User**(*key: str = ''*, *secret: str = ''*, *url: str = ''*, *proxy: str | None = None*)

Bases: *SpotClient*

Class that implements the Kraken Spot User client

Requires the Query funds permission in the API key settings.

Parameters

- **key** (*str, optional*) – Spot API public key (default: "")
- **secret** (*str, optional*) – Spot API secret key (default: "")
- **url** (*str, optional*) – The URL to access the Kraken API (default: https://api.kraken.com)
- **proxy** (*str, optional*) – proxy URL, may contain authentication information

Listing 1: Spot User: Create the user client

```
>>> from kraken.spot import User
>>> user = User() # unauthenticated
>>> auth_user = User(key="api-key", secret="secret-key") # authenticated
```

Listing 2: Spot User: Create the user client as context manager

```
>>> from kraken.spot import User
>>> with User(key="api-key", secret="secret-key") as user:
...     print(user.get_account_balances())
```

account_transfer(*asset: str*, *amount: str | float*, *from_: str*, *to_: str*, ***, *extra_params: dict | None = None*) → dict

Transfer funds between master and subaccounts. This is currently *only available for institutional clients and must be called by the master account*.

- https://docs.kraken.com/api/docs/rest-api/account-transfer

Parameters

- **asset** (*str*) – The asset to transfer
- **amount** (*str | float*) – The amount of that asset to transfer
- **from** (*str*) – The source account (IIBAN)
- **to** (*str*) – The destination account (IIBAN)

Returns
Success or failure

Return type
dict

Listing 3: Spot User: Transfer funds between accounts

```
>>> from kraken.spot import User
>>> user = User(key="api-key", secret="secret-key")
>>> user.account_transfer(
...     "asset": "XBT",
...     "amount": 1.0,
...     "from": "ABCD 1234 EFGH 5678"
...     "to": "IJKL 0987 MNOP 6543"
... )
{
    'result': {
        'transfer_id': 'TOH3AS2-LPCWR8-JDQGEU',
        'status': 'complete'
    }
}
```

create_subaccount(*username: str*, *email: str*, ***, *extra_params: dict | None = None*) → dict

Create a subaccount for trading. This is currently *only available for institutional clients*.

- https://docs.kraken.com/api/docs/rest-api/create-subaccount

Parameters

- **username** (*str*) – The username for the new subaccount
- **email** (*str*) – The E-Mail address for the new subaccount

Returns
Success or failure

Return type
dict

Listing 4: Spot User: Create a subaccount

```
>>> from kraken.spot import User
>>> user = User(key="api-key", secret="secret-key")
>>> user.create_subaccount(username="user", email="user@domain.com")
{ 'result': True }
```

delete_export_report(*id_: str*, *type_: str | None = 'delete'*, *, *extra_params: dict | None = None*) → dict

Delete a report from the Kraken server.

Requires the `Export data` permission. In addition for exporting trades data the permissions `Query open orders & trades` and `Query closed orders & trades` must be set. For exporting ledgers the `Query funds` and `Query ledger entries` must be set.

- https://docs.kraken.com/api/docs/rest-api/remove-export

Parameters

- **id** (*str*) – The id of the report
- **type** (*str, optional*) – The type of the export, one of: `cancel` and `delete` (default: `delete`)

Returns
Success or failure

Return type
dict

Listing 5: Spot User: Delete or cancel the report

```
>>> from kraken.spot import User
>>> user = User(key="api-key", secret="secret-key")
>>> user.delete_export_report(id_="GEHI", type_="delete")
{ 'delete': True }
```

get_account_balance(*, *extra_params: dict | None = None*) → dict

Get the current balances of the user.

Requires the `Query funds` permission in the API key settings.

- https://docs.kraken.com/api/docs/rest-api/get-account-balance

Listing 6: Spot User: Get the account balances

```
>>> from kraken.spot import User
>>> user = User(key="api-key", secret="secret-key")
>>> user.get_account_balances()
{
    'ZUSD': '241983.1415',
    'KFEE': '8020.22',
```

(continues on next page)

(continued from previous page)

```
    'BCH': '0.0000077100',
    'ETHW': '0.0000040',
    'XXLM': '0.00000000',
    'ZEUR': '0.0000',
    'DOT': '32011.21197000',
    ...
}
```

get_balance(*currency: str*) → dict

Returns the balance and available balance of a given currency.

Requires the `Query Funds` permission in the API key settings.

Parameters

currency (*str*) – The currency to get the balances from

Returns

Dictionary containing the `currency` (currency as string), `balance` (including value in orders), and `available_balance` (amount that is not in orders)

Return type

dict

Listing 7: Spot User: Get balance

```
>>> from kraken.spot import User
>>> user = User(key="api-key", secret="secret-key")
>>> user.get_balance(currency="EUR")
{
    'currency': 'ZEUR',
    'balance': 6011.2119,
    'available_balance': 4999.0619
}
```

get_balances(*, *extra_params: dict | None = None*) → dict

Retrieve the user's asset balances and the the corresponding amount held by open orders.

Requires the `Query Funds` permission in the API key settings.

- https://docs.kraken.com/api/docs/rest-api/get-extended-balance

Returns

Dictionary containing the `currency` as keys, that hold a dictionary containing the `balance` key holding the actual balance including the value in orders and the `hold_trade` key that represents the amount held in open orders.

Return type

dict

Listing 8: Spot User: Get balances

```
>>> from kraken.spot import User
>>> user = User(key="api-key", secret="secret-key")
>>> user.get_balances()
{
```

(continues on next page)

(continued from previous page)

```
    'XXLM': {
        'balance': '0.00000000', 'hold_trade': '0.00000000'
    },
    'ZEUR': {
        'balance': '500.0000', 'hold_trade': '0.0000'
    },
    'XXBT': {
        'balance': '2.1031709100', 'hold_trade': '0.1401000000'
    },
    'KFEE': {
        'balance': '1407.73', 'hold_trade': '0.00'
    },
    ...
}
```

get_closed_orders(*userref: int | None = None*, *start: int | None = None*, *end: int | None = None*, *ofs: int | None = None*, *closetime: str | None = 'both'*, ***, *trades: bool | None = False*, *extra_params: dict | None = None*) → dict

Get the 50 latest closed (filled or canceled) orders.

Requires the `Query closed orders & trades` permission in the API key settings.

- https://docs.kraken.com/api/docs/rest-api/get-closed-orders

Parameters

- **userref** (*int, optional*) – Filter the results by user reference id
- **start** (*int, optional*) – Unix timestamp to start the search from
- **end** (*int, optional*) – Unix timestamp to define the last result to include
- **ofs** (*int, optional*) – Offset for pagination
- **closetime** (*str, optional*) – Specify the exact time frame, one of: `both`, `open`, `close` (default: `both`)
- **trades** (*bool*) – Include trades related to position into the response or not (default: `False`)

Listing 9: Spot User: Get the closed orders

```
>>> from kraken.spot import User
>>> user = User(key="api-key", secret="secret-key")
>>> user.get_closed_orders()
{
    'closed': {
        'OBGFYP-XVQNL-P4GMWF': {
            'refid': None,
            'userref': 0,
            'status': 'closed',
            'opentm': 1680698929.9052045,
            'starttm': 0,
            'expiretm': 0,
            'descr': {
                'pair': 'ETHUSD',
```

(continues on next page)

(continued from previous page)

```
                'type': 'buy',
                'ordertype': 'limit',
                'price': '1860.76',
                'price2': '0',
                'leverage': 'none',
                'order': 'buy 0.02000000 ETHUSD @ limit 1860.76',
                'close': ''
            },
            'vol': '0.02000000',
            'vol_exec': '0.02000000',
            'cost': '37.21520',
            'fee': '0.05954',
            'price': '1860.76',
            'stopprice': '0.00000',
            'limitprice': '0.00000',
            'misc': '',
            'oflags': 'fciq',
            'reason': None,
            'closetm': 1680777419.8115675
        },
        'OAUHYR-YCVK6-P22G6P': {
            ...
        }
    }
}
```

get_export_report_status(*report: str*, *, *extra_params: dict | None = None*) → dict

Get the status of the current pending report.

Requires the `Export data` permission. In addition for exporting trades data the permissions `Query open orders & trades` and `Query closed orders & trades` must be set. For exporting ledgers the `Query funds` and `Query ledger entries` must be set.

- https://docs.kraken.com/api/docs/rest-api/export-status

Parameters
report (*str*) – Kind of report, one of: `trades`, `ledgers`

Returns
Information about the pending report

Return type
list[dict]

Listing 10: Example

```
>>> from kraken.spot import User
>>> user = User(key="api-key", secret="secret-key")
>>> response = user.request_export_report(
...     report="ledgers", description="myLedgers1", format="CSV"
... )
{ 'id': 'GEHI' }
>>> user.get_export_report_status(report="ledgers")
[
```

(continues on next page)

(continued from previous page)

```
    {
        'id': 'GEHI',
        'descr': 'myLedgers1',
        'format': 'CSV',
        'report': 'ledgers',
        'status': 'Queued',
        'aclass': 'currency',
        'fields': 'all',
        'asset': 'all',
        'subtype': 'all',
        'starttm': '1680307200',
        'endtm': '1680855267',
        'createdtm': '1680855267',
        'expiretm': '1682064867',
        'completedtm': '0',
        'datastarttm': '1680307200',
        'dataendtm': '1680855267',
        'flags': '0'
    }
]
```

get_ledgers(*id_: str | list[str]*, *, *trades: bool | None = False*, *extra_params: dict | None = None*) → dict

Get information about specific ledeger entries.

Requires the `Query funds` and `Query ledger entries` permissions in the API key settings.

- https://docs.kraken.com/api/docs/rest-api/get-ledgers-info

Parameters

- **id** (*str | list[str]*) – Ledger id as string, list of strings, or comma delimited list of ledger ids as string
- **trades** (*bool, optional*) – Include trades related to a position or not (default: `False`)

Listing 11: Spot User: Get ledgers

```
>>> from kraken.spot import User
>>> user = User(key="api-key", secret="secret-key")
>>> user.get_ledgers(id_="LKLSX7-VUXD4-HDLK2P")
{
    'LKLSX7-VUXD4-HDLK2P': {
        'aclass': 'currency',
        'amount': '0.00',
        'asset': 'FEE',
        'balance': '8020.22',
        'fee': '5.95',
        'refid': 'TPLJ5E-NONOU-5LH7JL',
        'time': 1680777419.8115911,
        'type': 'trade',
        'subtype': ''
    }
}
```

get_ledgers_info(*asset: str | list[str] | None = 'all'*, *aclass: str | None = 'currency'*, *type_: str | None = 'all'*, *start: int | None = None*, *end: int | None = None*, *ofs: int | None = None*, ***, *extra_params: dict | None = None*) → dict

Get information about the users ledger entries. 50 results can be returned at a time.

Requires the `Query funds` and `Query ledger entries` permissions in the API key settings.

- https://docs.kraken.com/api/docs/rest-api/get-ledgers

Parameters

- **asset** (*str | list[str]*) – The asset(s) to filter for (default: `all`)
- **aclass** (*str*) – The asset class (default: `currency`)
- **type** (*str, optional*) – Ledger type, one of: `all`, `deposit`, `withdrawal`, `trade`, `margin`, `rollover`, `credit`, `transfer`, `settled`, `staking`, and `sale` (default: `all`)
- **start** (*int, optional*) – Unix timestamp to start the search from
- **end** (*int, optional*) – Unix timestamp to define the last result
- **ofs** (*int, optional*) – Offset for pagination

Listing 12: Spot User: Get ledgers info

```
>>> from kraken.spot import User
>>> user = User(key="api-key", secret="secret-key")
>>> user.get_ledgers_info(asset=["KFEE","EUR","ETH"])
{
    'count': 519,
    'ledger': {
        'LKLSX7-VUXD4-HDLK2P': {
            'aclass': 'currency',
            'amount': '0.00',
            'asset': 'KFEE',
            'balance': '8020.22',
            'fee': '5.95',
            'refid': 'TPLJ5E-NONOU-5LH7JL',
            'time': 1680777419.8115911,
            'type': 'trade',
            'subtype': ''
        },
        'L4BF6E-FIFW7-6UB2CI': { ... },
        ...
    }
}
```

get_nonce() → str

Return a new nonce

get_open_orders(*userref: int | None = None*, ***, *trades: bool | None = False*, *extra_params: dict | None = None*) → dict

Get information about the open orders.

Requires the `Query open orders & trades` permission in the API key settings.

- https://docs.kraken.com/api/docs/rest-api/get-open-orders

Parameters

- **userref** (*int, optional*) – Filter the results by user reference id
- **trades** (*bool*) – Include trades related to position or not into the response (default: `False`)

Listing 13: Spot User: Get the open orders

```
>>> from kraken.spot import User
>>> user = User(key="api-key", secret="secret-key")
>>> user.get_open_orders()
{
    'open': {
        'OCUG7Z-4EM5R-7ZCJ47': {
            'refid': None,
            'userref': 0,
            'status':
            'open',
            'opentm': 1680777427.576083,
            'starttm': 0,
            'expiretm': 0,
            'descr': {
                'pair': 'ETHUSD',
                'type': 'buy',
                'ordertype': 'limit',
                'price': '1720.37',
                'price2': '0',
                'leverage': 'none',
                'order': 'buy 0.02000000 ETHUSD @ limit 1720.37',
                'close': ''
            },
            'vol': '0.02000000',
            'vol_exec': '0.00000000',
            'cost': '0.00000',
            'fee': '0.00000',
            'price': '0.00000',
            'stopprice': '0.00000',
            'limitprice': '0.00000',
            'misc': '',
            'oflags': 'fciq'
        },
        'OFZP3V-UMMUJ-6HMRMB': {
            ...
        }
    }
}
```

get_open_positions(*txid: str | list[str] | None = None*, *consolidation: str | None = 'market'*, ***, *docalcs: bool | None = False*, *extra_params: dict | None = None*) → dict

Get information about the open margin positions.

Requires the `Query open orders & trades` permission in the API key settings.

- https://docs.kraken.com/api/docs/rest-api/get-open-positions

Parameters

- **txid** (*str | list[str], optional*) – Filter by txid or list of txids or comma delimited list of txids as string
- **consolidation** (*str, optional*) – Consolidate positions by market/pair (default: `market`)
- **docalcs** (*bool, optional*) – Include profit and loss calculation into the result (default: `False`)

Returns
List of open positions

Return type
dict

Listing 14: Spot User: Get the open margin positions

```
>>> from kraken.spot import User
>>> user = User(key="api-key", secret="secret-key")
>>> user.get_open_positions()
{
    'TF5GVO-T7ZZ2-6NBKBI': {
        'ordertxid': 'O0SFFP-ABH4R-LOLNFG',
        'posstatus': 'open',
        'pair': 'XXBTZUSD',
        'time': 1618748097.12341,
        'type': 'buy',
        'ordertype': 'limit',
        'cost': '801243.52842',
        'fee': '208.44527',
        'vol': '8.82412861',
        'vol_closed': '0.20200000',
        'margin': '17234.123968',
        'value": '231463.1',
        'net": '+134186.9728',
        'terms": '0.0100% per 4 hours',
        'rollovertm': '1623672637',
        'misc': '',
        'oflags": ''
    }, ...
}
```

get_order_amends(*order_id: str*, *, *extra_params: dict | None = None*) → dict

Retrieve information about historical order amends.

Requires the `Query open orders & trades` and `Query closed orders & trades` permissions in the API key settings.

- https://docs.kraken.com/api/docs/rest-api/get-order-amends

Parameters
order_id – The `order_id` to filter for.

Listing 15: Spot User: Get order amends

```
>>> from kraken.spot import User
>>> user = User(key="api-key", secret="secret-key")
>>> user.get_order_amends(txid="OVM3PT-56ACO-53SM2T")
{'amends': [], 'count': 0}
```

get_orders_info(*txid: list[str] | str*, *userref: int | None = None*, ***, *trades: bool | None = False*, *consolidate_taker: bool | None = True*, *extra_params: dict | None = None*) → dict

Get information about one or more orders.

Requires the `Query open orders & trades` and `Query closed orders & trades` permissions in the API key settings.

- https://docs.kraken.com/api/docs/rest-api/get-orders-info

Parameters

- **txid** (*str | list[str]*) – A transaction id of a specific order, a list of txids or a string containing a comma delimited list of txids
- **userref** (*int, optional*) – Filter results by user reference id
- **trades** (*bool, optional*) – Include trades in the result or not (default: `False`)
- **consolidate_taker** (*bool, optional*) – Consolidate trades by individual taker trades (default: `True`)

Listing 16: Spot User: Get order information

```
>>> from kraken.spot import User
>>> user = User(key="api-key", secret="secret-key")
>>> user.get_orders_info(txid="OG5IL4-6AR7I-ZAPZEZ")
{
    'OG5IL4-6AR7I-ZAPZEZ': {
        'refid': None,
        'userref': 0,
        'status': 'open',
        'opentm': 1680618712.3723278,
        'starttm': 0,
        'expiretm': 0,
        'descr': {
            'pair': 'MATICUSD',
            'type': 'buy',
            'ordertype': 'limit',
            'price': '1.0922',
            'price2': '0',
            'leverage': 'none',
            'order': 'buy 45.77910000 MATICUSD @ limit 1.0922',
            'close': ''
        },
        'vol': '45.77910000',
        'vol_exec': '0.00000000',
        'cost': '0.000000',
        'fee': '0.000000',
```

(continues on next page)

(continued from previous page)

```
        'price': '0.000000',
        'stopprice': '0.000000',
        'limitprice': '0.000000',
        'misc': '',
        'oflags': 'fciq',
        'reason': None
    }
}
>>> user.get_orders_info(txid=["OAUHYR-YCVK6-P22G6P", "OG5IL4-6AR7I-ZAPZEZ"])
{
    'OAUHYR-YCVK6-P22G6P': {
        'refid': None,
        'userref': 0,
        'status': 'canceled',
        'opentm': 1680618716.4409518,
        'starttm': 0,
        'expiretm': 0,
        'descr': {
            'pair': 'MATICUSD',
            'type': 'buy',
            'ordertype': 'limit',
            'price': '1.0501',
            'price2': '0',
            'leverage': 'none',
            'order': 'buy 47.61450000 MATICUSD @ limit 1.0501',
            'close': ''
        },
        'vol': '47.61450000',
        'vol_exec': '0.00000000',
        'cost': '0.000000',
        'fee': '0.000000',
        'price': '0.000000',
        'stopprice': '0.000000',
        'limitprice': '0.000000',
        'misc': '',
        'oflags': 'fciq',
        'reason': 'User requested',
        'closetm': 1680756419.5768735
    }
}
```

get_trade_balance(*asset: str | None = 'ZUSD'*, *, *extra_params: dict | None = None*) → dict

Get the summary of all collateral balances.

Requires the `Query funds`, `Query open orders & trades`, and `Query closed orders & trades` permissions in the API key settings.

- https://docs.kraken.com/api/docs/rest-api/get-trade-balance

Parameters

asset (*str, optional*) – The base asset to determine the balances (default: ZUSD)

Listing 17: Spot User: Get the trade balance

```
>>> from kraken.spot import User
>>> user = User(key="api-key", secret="secret-key")
>>> user.get_trade_balance()
{
    'eb': '983691.5512', # Equivalent balance - all currencies combined
    'tb': '322296.9914', # Trade balance - balance of all equity currencies
    'm': '0.0000',       # Margin amount of open positions
    'uv': '0.0000',      # Unexecuted value of partly filled orders/positions
    'n': '0.0000',       # Unrealized net profit/loss of open positions
    'c': '0.0000',       # Cost basis of open positions
    'v': '0.0000',       # Current floating value of open positions
    'e': '983691.5512',  # Equity ( eb + n )
    'mf': '322296.9914'  # Free margin ( tb / initial margin ) * 100
}
```

get_trade_volume(*pair: str | list[str] | None = None*, **, fee_info: bool = True*, *extra_params: dict | None = None*) → dict

Get the 30-day user specific trading volume in USD.

Requires the `Query funds` permission in the API key settings.

- https://docs.kraken.com/api/docs/rest-api/get-trade-volume

Parameters

- **pair** (*str | list[str], optional*) – Asset pair, list of asset pairs or comma delimited list (as string) of asset pairs to filter
- **fee_info** (*bool, optional*) – Include fee information or not (default: `True`)

Listing 18: Spot User: Get the 30-day trade volume

```
>>> from kraken.spot import User
>>> user = User(key="api-key", secret="secret-key")
>>> user.get_trade_volume()
{
    'currency': 'ZUSD',
    'volume': '212220.9741',
    'fees': None,
    'fees_maker': None
}
>>> u.get_trade_volume(pair="DOTUSD")
{
    'currency': 'ZUSD',
    'volume': '212243.1210',
    'fees': {
        'DOTUSD': {
            'fee': '0.2200',
            'minfee': '0.1000',
            'maxfee': '0.2200',
            'nextfee': '0.2000',
            'tiervolume': '0.0000',
```

(continues on next page)

(continued from previous page)

```
            'nextvolume': '250000.0000'
        }
    },
    'fees_maker': {
        'DOTUSD': {
            'fee': '0.1200',
            'minfee': '0.0000',
            'maxfee': '0.1200',
            'nextfee': '0.1000',
            'tiervolume': '0.0000',
            'nextvolume': '250000.0000'
        }
    }
}
```

get_trades_history(*type_: str | None = 'all'*, *start: int | None = None*, *end: int | None = None*, *ofs: int | None = None*, *, trades: bool | None = False*, *consolidate_taker: bool = True*, *ledgers: bool = False*, *extra_params: dict | None = None*) → dict

Get information about the latest 50 trades and fills. Can be paginated.

Requires the `Query closed orders & trades` permission in the API key settings.

- https://docs.kraken.com/api/docs/rest-api/get-trade-history

Parameters

- **type** (*str, optional*) – Filter by type of trade, one of: `all`, `any position`, `closed position`, `closing position`, and `no position` (default: `all`)
- **start** (*int, optional*) – Timestamp or txid to start the search
- **end** (*int, optional*) – Timestamp or txid to define the last included result
- **trades** (*bool, optional*) – Include trades related to a position or not (default: `False`)
- **consolidate_taker** (*bool*) – Consolidate trades by individual taker trades (default: `True`)
- **ledgers** (*bool*) – Include related leger entries for filtered trade (default: `False`)

Listing 19: Spot User: Get the trade history

```
>>> from kraken.spot import User
>>> user = User(key="api-key", secret="secret-key")
>>> user.get_trades_history()
{
    'count': 630,
    'trades': {
        'TPLJ5E-NONOU-5LH7JL': {
            'ordertxid': 'OBGFYP-XVQNL-P4GMWF',
            'postxid': 'TKH2SE-M7IF5-CFI7LT',
            'pair': 'XETHZUSD',
            'time': 1680777419.8115635,
            'type': 'buy',
            'ordertype': 'limit',
```

(continues on next page)

(continued from previous page)

```
            'price': '1860.76000',
            'cost': '37.21520',
            'fee': '0.05954',
            'vol': '0.02000000',
            'margin': '0.00000',
            'leverage': '0',
            'misc': '',
            'trade_id': 43914718
        },
        'TNGMNU-XQSRA-LKCWOK': { ... },
        ...
    }
}
```

get_trades_info(*txid: str | list[str]*, *, *trades: bool | None = False*, *extra_params: dict | None = None*) → dict

Get information about specific trades/filled orders. 20 txids can be queried maximum.

Requires the `Query open orders & trades` and `Query closed orders & trades` permission in the API key settings.

- https://docs.kraken.com/api/docs/rest-api/get-trades-info

Parameters

- **txid** (*str | list[str]*) – txid or list of txids or comma delimited list of txids as string
- **trades** (*bool*) – Include trades related to position in result (default: `False`)

Listing 20: Spot User: Get the historical trade information

```
>>> from kraken.spot import User
>>> user = User(key="api-key", secret="secret-key")
>>> user.get_trades_info(txid="TNGMNU-XQSRA-LKCWOK")
{
    'TNGMNU-XQSRA-LKCWOK': {
        'ordertxid': 'OHAJCS-ON45W-UIXHT7',
        'postxid': 'TKH2SE-M7IF5-CFI7LT',
        'pair': 'XETHZUSD',
        'time': 1680606470.360982,
        'type': 'sell',
        'ordertype': 'limit',
        'price': '1855.16000',
        'cost': '37.10320',
        'fee': '0.05937',
        'vol': '0.02000000',
        'margin': '0.00000',
        'leverage': '0',
        'misc': '',
        'trade_id': 43878042
    }
}
```

request(*method: str*, *uri: str*, *params: dict | None = None*, *timeout: int = 10*, ***, *auth: bool = True*, *do_json: bool = False*, *return_raw: bool = False*, *query_str: str | None = None*, *extra_params: str | dict | None = None*) → dict[str, Any] | list[str] | list[dict[str, Any]] | Response

Handles the requested requests, by sending the request, handling the response, and returning the message or in case of an error the respective Exception.

Parameters

- **method** (*str*) – The request method, e.g., `GET`, `POST`, `PUT`, ...
- **uri** (*str*) – The endpoint to send the message
- **auth** (*bool*) – If the requests needs authentication (default: `True`)
- **params** (*dict, optional*) – The query or post parameter of the request (default: `None`)
- **extra_params** (*str | dict, optional*) – Additional query or post parameter of the request (default: `None`)
- **timeout** (*int*) – Timeout for the request (default: `10`)
- **do_json** (*bool*) – If the `params` must be "jsonified" - in case of nested dict style
- **return_raw** (*bool, optional*) – If the response should be returned without parsing. This is used for example when requesting an export of the trade history as .zip archive.
- **query_str** (*str, optional*) – Add custom values to the query /0/public/Nfts?filter%5Bcollection_id%5D=NCQNABO-XYCA7-JMMSDF&page_size=10

Raises

kraken.exceptions.KrakenException.* – If the response contains errors

Returns

The response

Return type

dict | list | requests.Response

request_export_report(*report: str*, *description: str*, *format_: str | None = 'CSV'*, *fields: str | list[str] | None = 'all'*, *starttm: int | None = None*, *endtm: int | None = None*, ***, *timeout: int | None = 10*, *extra_params: dict | None = None*) → dict

Request to export the trades or ledgers of the user.

Requires the `Export data` permission. In addition for exporting trades data the permissions `Query open orders & trades` and `Query closed orders & trades` must be set. For exporting ledgers the `Query funds` and `Query ledger entries` must be set.

- https://docs.kraken.com/api/docs/rest-api/add-export

Parameters

- **report** (*str*) – Kind of report, one of: `trades` and `ledgers`
- **format** (*str*) – The export format of the requesting report, one of `CSV` and `TSV` (default: `CSV`)
- **fields** (*str | list[str], optional*) – Fields to include in the report (default: `all`)
- **starttm** (*int, optional*) – Unix timestamp to start
- **endtm** (*int, optional*) – Unix timestamp of the last result
- **timeout** (*int*) – The timeout for that request (default: `10`)

Returns
A dictionary containing the export id

Return type
dict

Listing 21: Spot User: Request an report export

```
>>> from kraken.spot import User
>>> user = User(key="api-key", secret="secret-key")
>>> user.request_export_report(
...     report="ledgers", description="myLedgers1", format="CSV"
... )
{ 'id': 'GEHI' }
```

retrieve_export(*id_: str*, *timeout: int = 10*, ***, *extra_params: dict | None = None*) → dict

Retrieve the requested report export.

Requires the `Export data` permission. In addition for exporting trades data the permissions `Query open orders & trades` and `Query closed orders & trades` must be set. For exporting ledgers the `Query funds` and `Query ledger entries` must be set.

- https://docs.kraken.com/api/docs/rest-api/retrieve-export

Parameters

- **id** (*str*) – Id of the report that was requested
- **timeout** (*int, optional*) – Timeout for that request, default: `10` seconds

Returns
The response - a zipped report

Return type
requests.Response

Listing 22: Spot User: Save the exported report to CSV

```
>>> from kraken.spot import User
>>> user = User(key="api-key", secret="secret-key")
>>> response = user.request_export_report(
...     report="ledgers", description="myLedgers1", format="CSV"
... )
{ 'id': 'GEHI' }
>>> ledgers_data = user.retrieve_export(id_=response["id"])
>>> with open("myExport.zip", "wb") as file:
...     for chunk in ledgers_data.iter_content(chunk_size=512):
...         if chunk:
...             file.write(chunk)
```

property return_unique_id: str

Returns a unique uuid string

Returns
uuid

Return type
str

class kraken.spot.**Market**(*key: str = ''*, *secret: str = ''*, *url: str = ''*, *proxy: str | None = None*)

Bases: *SpotClient*

Class that implements the Kraken Spot Market client. Can be used to access the Kraken Spot market data.

Parameters

- **key** (*str, optional*) – Spot API public key (default: "")
- **secret** (*str, optional*) – Spot API secret key (default: "")
- **url** (*str, optional*) – Alternative URL to access the Kraken API (default: https://api.kraken.com)
- **proxy** (*str, optional*) – proxy URL, may contain authentication information

Listing 23: Spot Market: Create the market client

```
>>> from kraken.spot import Market
>>> market = Market() # unauthenticated
>>> auth_market = Market(key="api-key", secret="secret-key") # authenticated
```

Listing 24: Spot Market: Create the market client as context manager

```
>>> from kraken.spot import Market
>>> with Market() as market:
...     print(market.get_assets())
```

get_asset_pairs(*pair: str | list[str] | None = None*, *info: str | None = None*, *aclass_base: str | None = None*, *, *extra_params: dict | None = None*) → dict

Get information about a single or multiple asset/currency pair(s). If `pair` is left blank, all currency pairs will be returned.

- https://docs.kraken.com/api/docs/rest-api/get-tradable-asset-pairs

This function uses caching. Run `get_asset_pairs.cache_clear()` to clear.

Parameters

- **pair** (*str | list[str], optional*) – Filter by asset pair(s)
- **info** (*str, optional*) – Filter by info, can be one of: `info` (all info), `leverage` (leverage info), `fees` (fee info), and `margin` (margin info)

Returns
Information about the asset pair

Return type
dict

Listing 25: Spot Market: Get information about tradeable asset pairs

```
>>> from kraken.spot import Market
>>> Market().get_asset_pairs(pair="XBTUSD")
{
    'XXBTZUSD': {
        'altname': 'XBTUSD',
        'wsname': 'XBT/USD',
        'aclass_base': 'currency',
        'base': 'XXBT',
```

(continues on next page)

(continued from previous page)

```
        'aclass_quote': 'currency',
        'quote': 'ZUSD',
        'lot': 'unit',
        'cost_decimals': 5,
        'pair_decimals': 1,
        'lot_decimals': 8,
        'lot_multiplier': 1,
        'leverage_buy': [2, 3, 4, 5],
        'leverage_sell': [2, 3, 4, 5],
        'fees': [
            [0, 0.26], [50000, 0.24], [100000, 0.22],
            [250000, 0.2], [500000, 0.18], [1000000, 0.16],
            [2500000, 0.14], [5000000, 0.12], [10000000, 0.1]
        ],
        'fees_maker': [
            [0, 0.16], [50000, 0.14], [100000, 0.12],
            [250000, 0.1], [500000, 0.08], [1000000, 0.06],
            [2500000, 0.04], [5000000, 0.02], [10000000, 0.0]
        ],
        'fee_volume_currency': 'ZUSD',
        'margin_call': 80,
        'margin_stop': 40,
        'ordermin': '0.0001',
        'costmin': '0.5',
        'tick_size': '0.1',
        'status': 'online',
        'long_position_limit': 270,
        'short_position_limit': 180
    }
}
```

get_assets(*assets: str | list[str] | None = None*, *aclass: str | None = None*, ***, *extra_params: dict | None = None*) → dict

Get information about one or more assets. If `assets` is not specified, all assets will be returned.

- https://docs.kraken.com/api/docs/rest-api/get-asset-info

This function uses caching. Run `get_assets.cache_clear()` to clear.

Parameters

- **assets** (*str | list[str], optional*) – Filter by asset(s)
- **aclass** (*str, optional*) – Filter by asset class

Returns

Information about the requested assets

Return type

dict

Listing 26: Spot Market: Get information about the available assets

```
>>> from kraken.spot import Market
>>> market = Market()
```

(continues on next page)

(continued from previous page)

```
>>> market.get_assets(assets="DOT")
{
    'DOT': {
        'aclass': 'currency',
        'altname': 'DOT',
        'decimals': 10,
        'display_decimals': 8,
        'collateral_value': 0.9,
        'status': 'enabled'
    }
}
>>> market.get_assets(assets=["MATIC", "XBT"]) # same as market.get_
↪assets(assets="MATIC,XBT"])
    'MATIC': {
        'aclass': 'currency',
        'altname': 'MATIC',
        'decimals': 10,
        'display_decimals': 5,
        'collateral_value': 0.7,
        'status': 'enabled'
    },
    'XXBT': {
        'aclass': 'currency',
        'altname': 'XBT',
        'decimals': 10,
        'display_decimals': 5,
        'collateral_value': 1.0,
        'status': 'enabled'
    }
}
```

get_nonce() → str

Return a new nonce

get_ohlc(*pair: str*, *interval: int | str = 1*, *since: int | str | None = None*, ***, *extra_params: dict | None = None*) → dict

Get the open, high, low, and close data for a specific trading pair. Returns at max 720 time steps per request.

- https://docs.kraken.com/api/docs/rest-api/get-ohlc-data

Parameters

- **pair** (*str*) – The pair to get the ohlc from
- **interval** (*str | int, optional*) – the Interval in minutes (default: 1)
- **since** (*int | str, optional*) – Timestamp to start from (default: None)

Returns

The OHLC data of a given asset pair

Return type

dict

Listing 27: Spot Market: Get the OHLC data

```
>>> from kraken.spot import Market
>>> Market().get_ohlc(pair="XBTUSD")
{
    "XXBTZUSD": [
        [
            1680671100,
            "28488.9",
            "28489.0",
            "28488.8",
            "28489.0",
            "28488.9",
            "1.03390376",
            8
        ], ...
    ]
}
```

get_order_book(*pair: str*, *count: int | None = 100*, ***, *extra_params: dict | None = None*) → dict

Get the current orderbook of a specified trading pair.

- https://docs.kraken.com/api/docs/rest-api/get-order-book

Parameters

- **pair** (*str*) – The pair to get the orderbook from
- **count** (*int, optional*) – Number of asks and bids, must be one of {1..500} (default: 100)

Returns

Return type
dict

Listing 28: Spot Market: Get the orderbook

```
>>> from kraken.spot import Market
>>> Market().get_order_book(pair="XBTUSD", count=2)
{
    'XXBTZUSD': {
        'asks': [
            ['28000.00000', '1.091', 1680714417],
            ['28001.00000', '0.001', 1680714413]
        ],
        'bids': [
            ['27999.90000', '2.240', 1680714419],
            ['27999.50000', '0.090', 1680714418]
        ]
    }
}
```

get_recent_spreads(*pair: str*, *since: str | int | None = None*, ***, *extra_params: dict | None = None*) → dict

Get the latest spreads for a specific trading pair.

- https://docs.kraken.com/api/docs/rest-api/get-recent-spreads

Parameters

- **pair** (*str*) – Pair to get the recent spreads
- **since** (*str | int, optional*) – Filter trades since given timestamp (default: None)

Returns
The last *n* spreads of the asset pair

Return type
dict

Listing 29: Spot Market: Get the recent spreads

```
>>> from kraken.spot import Market
>>> Market().get_recent_spreads(pair="XBTUSD")
{
    "XXBTZUSD": [
        [1680714601, "28015.00000", "28019.40000"],
        [1680714601, "28015.00000", "28017.00000"],
        [1680714601, "28015.00000", "28016.90000"],
        ...
    ]
}
```

get_recent_trades(*pair: str*, *since: str | int | None = None*, *count: int | None = None*, ***, *extra_params: dict | None = None*) → dict

Get the latest trades for a specific trading pair (up to 1000).

- https://docs.kraken.com/api/docs/rest-api/get-recent-trades

Parameters

- **pair** (*str*) – Pair to get the recent trades
- **since** (*str | int, optional*) – Filter trades since given timestamp (default: None)
- **count** (*int, optional*) – The max number of results

Returns
The last public trades (up to 1000 results)

Return type
dict

Listing 30: Spot Market: Get the recent trades

```
>>> from kraken.spot import Market
>>> Market().get_recent_trades(pair="XBTUSD")
{
    "XXBTZUSD": [
        ["27980.90000", "0.00071054", 1680712703.2524643, "b", "l", "",␣
↪57811127],
        ["27981.00000", "0.03180000", 1680712715.1806278, "b", "l", "",␣
↪57811128],
        ["27980.90000", "0.00010000", 1680712715.469506, "s", "m", "",␣
```

(continues on next page)

(continued from previous page)

```
    ↪57811129],
            ...
        ]
}
```

get_system_status(*, *extra_params: dict | None = None*) → dict

Returns the system status of the Kraken Spot API.

- https://docs.kraken.com/api/docs/rest-api/get-system-status

Returns
Success or failure

Return type
dict

Listing 31: Spot Market: Get the Kraken API system status

```
>>> from kraken.spot import Market
>>> Market().get_system_status()
{'status': 'online', 'timestamp': '2023-04-05T17:12:31Z'}
```

get_ticker(*pair: str | list[str] | None = None*, *, *extra_params: dict | None = None*) → dict

Returns all tickers if pair is not specified - else just the ticker of the `pair`. Multiple pairs can be specified.

- https://docs.kraken.com/api/docs/rest-api/get-ticker-information

Parameters
pair (*str | list[str], optional*) – Filter by pair(s)

Returns
The ticker(s) including ask, bid, close, volume, vwap, high, low, todays open and more

Return type
dict

Listing 32: Spot Market: Get the ticker(s)

```
>>> from kraken.spot import Market
>>> Market().get_ticker(pair="XBTUSD")
{
    'XXBTZUSD': {
        'a': ['27948.00000', '3', '3.000'],      # ask
        'b': ['27947.90000', '1', '1.000'],      # bid
        'c': ['27947.90000', '0.00842808'],      # last trade close, lot volume
        'v': ['3564.58017484', '4138.93906134'], # volume today and last 24h
        'p': ['28351.31431', '28329.55480'],     # vwap today and last 24h
        't': [33574, 43062],                     # number of trades today and␣
    ↪last 24h
        'l': ['27813.10000', '27813.10000'],     # low today and last 24h
        'h': ['28792.30000', '28792.30000'],     # high today and last 24
        'o': '28173.00000'                       # today's opening price
    }
}
```

request(*method: str*, *uri: str*, *params: dict | None = None*, *timeout: int = 10*, ***, *auth: bool = True*, *do_json: bool = False*, *return_raw: bool = False*, *query_str: str | None = None*, *extra_params: str | dict | None = None*) → dict[str, Any] | list[str] | list[dict[str, Any]] | Response

Handles the requested requests, by sending the request, handling the response, and returning the message or in case of an error the respective Exception.

Parameters

- **method** (*str*) – The request method, e.g., `GET`, `POST`, `PUT`, ...
- **uri** (*str*) – The endpoint to send the message
- **auth** (*bool*) – If the requests needs authentication (default: `True`)
- **params** (*dict, optional*) – The query or post parameter of the request (default: `None`)
- **extra_params** (*str | dict, optional*) – Additional query or post parameter of the request (default: `None`)
- **timeout** (*int*) – Timeout for the request (default: `10`)
- **do_json** (*bool*) – If the `params` must be "jsonified" - in case of nested dict style
- **return_raw** (*bool, optional*) – If the response should be returned without parsing. This is used for example when requesting an export of the trade history as .zip archive.
- **query_str** (*str, optional*) – Add custom values to the query /0/public/Nfts?filter%5Bcollection_id%5D=NCQNABO-XYCA7-JMMSDF&page_size=10

Raises

kraken.exceptions.KrakenException.* – If the response contains errors

Returns

The response

Return type

dict | list | requests.Response

property return_unique_id: str

Returns a unique uuid string

Returns

uuid

Return type

str

class kraken.spot.**Trade**(*key: str = ''*, *secret: str = ''*, *url: str = ''*, *proxy: str | None = None*)

Bases: *SpotClient*

Class that implements the Kraken Trade Spot client.

Parameters

- **key** (*str, optional*) – Spot API public key (default: `""`)
- **secret** (*str, optional*) – Spot API secret key (default: `""`)
- **url** (*str, optional*) – The URL to access the Kraken API (default: https://api.kraken.com)
- **proxy** (*str, optional*) – proxy URL, may contain authentication information

Listing 33: Spot Trade: Create the trade client

```
>>> from kraken.spot import Trade
>>> trade = Trade() # unauthenticated
>>> auth_trade = Trade(key="api-key", secret="secret-key") # authenticated
```

Listing 34: Spot Trade: Create the trade client as context manager

```
>>> from kraken.spot import Trade
>>> with Trade(key="api-key", secret="secret-key") as trade:
...     print(trade.create_order(...))
```

amend_order(*, *extra_params: dict | None = None*) → dict

Amend/modify an open order.

Requires the `Create and modify orders` and `Cancel & close orders` permissions in the API key settings.

- https://docs.kraken.com/api/docs/rest-api/amend-order

Listing 35: Spot Trade: Amend order

```
>>> from kraken.spot import Trade
>>> trade = Trade(key="api-key", secret="secret-key")
>>> trade.amend_order(
...     extra_params={
...         "txid": "OVM3PT-56ACO-53SM2T",
...         "limit_price": "105636.9",
...     }
... )
```

cancel_all_orders(*, *extra_params: dict | None = None*) → dict

Cancel all open orders.

Requires the `Cancel/close orders` permission in the API key settings.

- https://docs.kraken.com/api/docs/rest-api/cancel-all-orders

Returns
Success or failure - Number of closed orders

Return type
dict

Listing 36: Spot Trade: Cancel all open orders

```
>>> from kraken.spot import Trade
>>> trade = Trade(key="api-key", secret="secret-key")
>>> trade.cancel_all_orders()
{ 'count': 2 }
```

cancel_all_orders_after_x(*timeout: int = 0*, *, *extra_params: dict | None = None*) → dict

Cancel all orders after a timeout. This can be used as Dead Man's Switch.

Requires the `Create and modify orders` permission in the API key settings.

- https://docs.kraken.com/api/docs/rest-api/cancel-all-orders-after

Parameters
timeout (*int, optional*) – Optional The timeout in seconds, deactivate by passing the default: `0`

Returns
Current time and trigger time

Return type
dict

Listing 37: Spot Trade: Set the Death Man's Switch

```
>>> from kraken.spot import Trade
>>> trade = Trade(key="api-key", secret="secret-key")
>>> trade.cancel_all_orders_after_x(timeout=60)
{
    'currentTime': '2023-04-06T06:51:56Z',
    'triggerTime': '2023-04-06T06:52:56Z'
}
```

cancel_order(*txid: str*, *, *extra_params: dict | None = None*) → dict

Cancel a specific order by `txid`. Instead of a transaction id a user reference id can be passed.

Requires the `Cancel/close orders` permission in the API key settings.

- https://docs.kraken.com/api/docs/rest-api/cancel-order

Parameters
txid (*str*) – Transaction id or comma delimited list of user reference ids to cancel.

Returns
Success or failure - Number of closed orders

Return type
dict

Listing 38: Spot Trade: Cancel an order

```
>>> from kraken.spot import Trade
>>> trade = Trade(key="api-key", secret="secret-key")
>>> trade.cancel_order(txid="OAUHYR-YCVK6-P22G6P")
{ 'count': 1 }
```

cancel_order_batch(*orders: list[str | int]*, *, *extra_params: dict | None = None*) → dict

Cancel a a list of orders by `txid` or `userref`

Requires the `Cancel/close orders` permission in the API key settings.

- https://docs.kraken.com/api/docs/rest-api/cancel-order-batch

Parameters
orders (*list[str | int]*) – List of orders to cancel

Returns
Success or failure - Number of closed orders

Return type
dict

Listing 39: Spot Trade: Cancel multiple orders

```
>>> from kraken.spot import Trade
>>> trade = Trade(key="api-key", secret="secret-key")
>>> trade.cancel_order_batch(
...     orders=["OG5IL4-6AR7I-ZAPZEZ", "OAUHYR-YCVK6-P22G6P"]
... )
{ count': 2 }
```

create_order(*ordertype: str*, *side: str*, *pair: str*, *volume: str | float*, *price: str | float | None = None*, *price2: str | float | None = None*, *trigger: str | None = None*, *leverage: str | None = None*, *stptype: str | None = 'cancel-newest'*, *oflags: str | list[str] | None = None*, *timeinforce: str | None = None*, *displayvol: str | None = None*, *starttm: str | None = '0'*, *expiretm: str | None = None*, *close_ordertype: str | None = None*, *close_price: str | float | None = None*, *close_price2: str | float | None = None*, *deadline: str | None = None*, *userref: int | None = None*, ***, *truncate: bool = False*, *reduce_only: bool | None = False*, *validate: bool = False*, *extra_params: dict | None = None*) → dict

Create a new order and place it on the market.

Requires the `Create and modify orders` permission in the API key settings.

If the traded asset is a tokenized asset, *extra_params* must contain the key *"asset_class"* with value *"tokenized_asset"*.

- https://docs.kraken.com/api/docs/rest-api/add-order

Parameters

- **ordertype** (*str*) – The kind of the order, one of: `market`, `limit`, `take-profit`, `stop-loss-limit`, `take-profit-limit` and `settle-position` (see: https://support.kraken.com/hc/en-us/sections/200577136-Order-types)
- **side** (*str*) – `buy` or `sell`
- **pair** (*str*) – The asset to trade
- **volume** (*str | float*) – The volume of the position to create
- **price** (*str | float, optional*) – The limit price for `limit` orders and the trigger price for orders with `ordertype` one of `stop-loss`, `stop-loss-limit`, `take-profit`, and `take-profit-limit`
- **price2** (*str | float, optional*) – The limit price for `stop-loss-limit` and `take-profit-limit` orders The price2 can also be set to absolute or relative changes.
  - Prefixed using + or - defines the change in the quote asset
  - Prefixed by # is the same as + or - but the sign is set automatically
  - The percentage sign (%) can be used to define relative changes.
- **trigger** (*str, optional*) – What triggers the position of `stop-loss`, `stop-loss-limit`, `take-profit`, and `take-profit-limit` orders. Will also be used for associated conditional close orders. Kraken will use `last` if nothing is specified.
- **leverage** (*str | float, optional*) – The leverage

- **stptype** (*str, optional*) – Define what cancels the order, one of `cancel-newest`, `cancel-oldest`, `cancel-both` (default: `cancel-newest`)
- **oflags** (*str | list[str], optional*) – Order flags like `post`, `fcib`, `fciq`, `nomp`, `viqc` (see the referenced Kraken documentation for more information)
- **timeinforce** (*str, optional*) – how long the order remains in the orderbook, one of: `GTC`, `IOC`, `GTD` (see the referenced Kraken documentation for more information)
- **displayvol** (*str | float, optional*) – Define how much of the volume is visible in the orderbook (iceberg)
- **starttim** (*str, optional*) – Unix timestamp or seconds defining the start time (default: "0")
- **expiretm** (*str, optional*) – Unix timestamp or time in seconds defining the expiration of the order, (default: "0" - i.e., no expiration)
- **close_ordertype** (*str, optional*) – Conditional close order type, one of: `limit`, `stop-loss`, `take-profit`, `stop-loss-limit`, `take-profit-limit` (see the referenced Kraken documentation for more information)
- **close_price** (*str | float, optional*) – Conditional close price
- **close_price2** (*str | float, optional*) – The price2 for the conditional order - see the price2 parameter description
- **deadline** (*str, optional*) – RFC3339 timestamp + {0..60} seconds that defines when the matching engine should reject the order.
- **truncate** (*bool, optional*) – If enabled: round the `price` and `volume` to Kraken's maximum allowed decimal places. See https://support.kraken.com/hc/en-us/articles/4521313131540 fore more information about decimals.
- **reduce_only** (*bool, optional*) – Reduce existing orders (default: `False`)
- **validate** (*bool, optional*) – Validate the order without placing on the market (default: `False`)
- **userref** (*int, optional*) – User reference id for example to group orders

Raises
ValueError – If input is not correct

Returns
The transaction id

Return type
dict

Listing 40: Spot Trade: Create a market order

```
>>> from kraken.spot import Trade
>>> trade = Trade(key="api-key", secret="secret-key")
>>> trade.create_order(
...     ordertype="market",
...     side="buy",
...     pair="XBTUSD",
...     volume="0.0001"
... )
{
```

(continues on next page)

(continued from previous page)

```
    'txid': ['TNGMNU-XQSRA-LKCWOK'],
    'descr': {
        'order': 'buy 4.00000000 XBTUSD @ limit 23000.0'
    }
}
```

Listing 41: Spot Trade: Create limit order

```
>>> from kraken.spot import Trade
>>> trade = Trade(key="api-key", secret="secret-key")
>>> trade.create_order(
...     ordertype="limit",
...     side="buy",
...     pair="XBTUSD",
...     volume=4,
...     price=23000,
...     expiretm=121,
...     displayvol=0.5,
...     oflags=["post", "fcib"]
... )
{
    'txid': ['TPPI2H-CUZZ2-EQR2IE'],
    'descr': {
        'order': 'buy 4.0000 XBTUSD @ limit 23000.0'
    }
}
```

Listing 42: Spot Trade: Create a stop loss order

```
>>> from kraken.spot import Trade
>>> trade = Trade(key="api-key", secret="secret-key")
>>> trade.create_order(
...     ordertype="stop-loss",
...     pair="XBTUSD",
...     volume=20,
...     price=22000,
...     side="buy",
... )
{
    'txid': ['THNUL1-8ZAS5-EEF3A8'],
    'descr': {
        'order': 'buy 20.00000000 XBTUSD @ stop loss 22000.0'
    }
}
```

Listing 43: Spot Trade: Create stop-loss-limit and take-profit-limit orders

```
''' When the price hits $25000:
   1. A limit buy order will be placed at $24000 with 2x leverage.
   2. When the limit order gets closed/filled at $24000 The
      stop-loss-limit part is done and the tale-profit-limit part
      begins.
```

(continues on next page)

(continued from previous page)

```
   3. When the price hits $27000 a limit order will be placed at
      $26800 to sell 1.2 BTC. This ensures that the asset will be
      sold for $26800 or better.
'''

>>> from kraken.spot import Trade
>>> trade = Trade(key="api-key", secret="secret-key")
>>> from datetime import datetime, timedelta, timezone
>>> deadline = (
...     datetime.now(timezone.utc) + timedelta(seconds=20)
... ).isoformat()
>>> trade.create_order(
...     ordertype="stop-loss-limit",
...     pair="XBTUSD",
...     side="buy",
...     volume=1.2,
...     price=24000,
...     price2=25000,
...     validate=True, # just validate the input, do not place on the market
...     trigger="last",
...     timeinforce="GTC",
...     leverage=4,
...     deadline=deadline,
...     close_ordertype="take-profit-limit",
...     close_price=27000,
...     close_price2=26800,
... )
{
    'descr': {
        'order': 'buy 0.00100000 XBTUSD @ stop loss 24000.0 -> limit
        25000.0 with 2:1 leverage', 'close': 'close position @ take
        profit 27000.0 -> limit 26800.0'
    }
}

''' The price2 and close_price2 can also be set to absolute or
relative changes.
    * Prefixed using "+" or "-" defines the change in the quote
      asset
    * Prefixed by # is the same as "+" and "-" but the sign is set
      automatically
    * The the percentage sign "%" can be used to define relative
      changes.
'''
>>> trade.create_order(
...     ordertype="stop-loss-limit",
...     pair="XBTUSD",
...     side="buy",
...     volume=1.2,
...     price=24000,
...     price2="+1000",
...     validate=True,
```

(continues on next page)

(continued from previous page)

```
...     trigger="last",
...     timeinforce="GTC",
...     close_ordertype="take-profit-limit",
...     close_price=27000,
...     close_price2="#2%",
... )
{
    'descr': {
        'order': 'buy 0.00100000 XBTUSD @ stop loss 24000.0 -> limit
        +1000.0', 'close': 'close position @ take profit 27000.0 ->
        limit -2.0000%'
    }
}
```

create_order_batch(*orders: list[dict]*, *pair: str*, *deadline: str | None = None*, ***, *validate: bool = False*, *extra_params: dict | None = None*) → dict

Create a batch of max 15 orders for a specific asset pair.

Requires the `Create and modify orders` permission in the API key settings.

- https://docs.kraken.com/api/docs/rest-api/add-order-batch

Parameters

- **orders** (*list[dict]*) – Dictionary of order objects (see the referenced Kraken documentation for more information)
- **pair** (*str*) – Asset pair to place the orders for
- **deadline** (*str, optional*) – RFC3339 timestamp + {0..60} seconds that defines when the matching engine should reject the order.
- **validate** (*bool, optional*) – Validate the orders without placing them. (default: `False`)

Returns
Information about the placed orders

Return type
dict

Listing 44: Spot Trade: Create a batch order

```
>>> from kraken.spot import Trade
>>> trade = Trade(key="api-key", secret="secret-key")
>>> trade.create_order_batch(orders=[
...         {
...             "close": {
...                 "ordertype": "stop-loss-limit",
...                 "price": 21000,
...                 "price2": 20000,
...             },
...             "ordertype": "limit",
...             "price": 25000,
...             "timeinforce": "GTC",
```

(continues on next page)

(continued from previous page)

```
...             "type": "buy",
...             "userref": 16861348843,
...             "volume": 1,
...         },
...         {
...             "ordertype": "limit",
...             "price": 1000000,
...             "timeinforce": "GTC",
...             "type": "sell",
...             "userref": 16861348843,
...             "volume": 2,
...         },
...     ],
...     pair="BTC/USD"
... )
{
    'orders': [{
        'order': 'buy 1 BTCUSD @ limit 25000',
        'txid': 'O5TLGX-DKKTU-WKRAZ5',
        'close': 'close position @ stop loss 21000.0 -> limit 20000.0'
    }, {
        'order': "sell 2 BTCUSD @ limit 1000000',
        'txid': 'OBGFYP-XVQNL-P4GMWF'
    }]
}
```

edit_order(*txid: str*, *pair: str*, *volume: str | float | None = None*, *price: str | float | None = None*, *price2: str | float | None = None*, *oflags: str | None = None*, *deadline: str | None = None*, *cancel_response: bool | None = None*, *userref: int | None = None*, ***, *truncate: bool = False*, *validate: bool = False*, *extra_params: dict | None = None*) → dict

Edit an open order.

Requires the `Create and modify orders` permission in the API key settings.

- https://docs.kraken.com/api/docs/rest-api/edit-order

Parameters

- **txid** (*str*) – The txid of the order to edit
- **pair** (*str*) – The asset pair of the order
- **volume** (*str | float, optional*) – Set a new volume
- **price** (*str | float, optional*) – Set a new price
- **price2** (*str | float, optional*) – Set a new second price
- **oflags** (*str | list[str], optional*) – Order flags like `post`, `fcib`, `fciq`, `nomp`, `viqc` (see the referenced Kraken documentation for more information)
- **deadline** (*string*) – (see the referenced Kraken documentation for more information)
- **cancel_response** (*bool, optional*) – See the referenced Kraken documentation for more information

- **truncate** (*bool, optional*) – If enabled: round the `price` and `volume` to Kraken's maximum allowed decimal places. See https://support.kraken.com/hc/en-us/articles/4521313131540 fore more information about decimals.
- **validate** (*bool, optional*) – Validate the order without placing on the market (default: `False`)
- **userref** (*int*) – User reference id for example to group orders

Returns
Success or failure

Return type
dict

Listing 45: Spot Trade: Modify an order

```
>>> from kraken.spot import Trade
>>> trade = Trade(key="api-key", secret="secret-key")
>>> trade.edit_order(txid="OBGFYP-XVQNL-P4GMWF",
...     volume=0.75,
...     pair="XBTUSD",
...     price=1250000
... )
{
    'status': 'ok',
    'txid': 'OFVXHJ-KPQ3B-VS7ELA',
    'originaltxid': 'OBGFYP-XVQNL-P4GMWF',
    'volume': '0.75',
    'price': '1250000',
    'orders_cancelled': 1,
    'descr': {
        'order': 'sell 0.75 XXBTZUSD @ limit 1250000'
    }
}
```

get_nonce() → str

Return a new nonce

request(*method: str*, *uri: str*, *params: dict | None = None*, *timeout: int = 10*, ***, *auth: bool = True*, *do_json: bool = False*, *return_raw: bool = False*, *query_str: str | None = None*, *extra_params: str | dict | None = None*) → dict[str, Any] | list[str] | list[dict[str, Any]] | Response

Handles the requested requests, by sending the request, handling the response, and returning the message or in case of an error the respective Exception.

Parameters

- **method** (*str*) – The request method, e.g., `GET`, `POST`, `PUT`, ...
- **uri** (*str*) – The endpoint to send the message
- **auth** (*bool*) – If the requests needs authentication (default: `True`)
- **params** (*dict, optional*) – The query or post parameter of the request (default: `None`)
- **extra_params** (*str | dict, optional*) – Additional query or post parameter of the request (default: `None`)
- **timeout** (*int*) – Timeout for the request (default: `10`)

- **do_json** (*bool*) – If the params must be "jsonified" - in case of nested dict style
- **return_raw** (*bool, optional*) – If the response should be returned without parsing. This is used for example when requesting an export of the trade history as .zip archive.
- **query_str** (*str, optional*) – Add custom values to the query /0/public/Nfts?filter%5Bcollection_id%5D=NCQNABO-XYCA7-JMMSDF&page_size=10

Raises
kraken.exceptions.KrakenException.* – If the response contains errors

Returns
The response

Return type
dict | list | requests.Response

property return_unique_id: str

Returns a unique uuid string

Returns
uuid

Return type
str

truncate(*amount: Decimal | float | str*, *amount_type: str*, *pair: str*, *asset_class: str = 'currency'*) → str

Kraken only allows volume and price amounts to be specified with a specific number of decimal places, and these vary depending on the currency pair used.

This function converts an amount of a specific type and pair to a string that uses the correct number of decimal places.

This function uses caching. Run `truncate.clear_cache()` to clear.

Parameters

- **amount** (*Decimal | float | str*) – The floating point number to represent
- **amount_type** (*str*) – What the amount represents. Either "price" or "volume"
- **pair** (*str*) – The currency pair the amount is in reference to.
- **asset_class** (*str, optional*) – The asset class of the base currency. Default is "currency". If the traded asset is a tokenized asset, set this to "tokenized_asset".

Raises

- **ValueError** – If the amount_type is price and the price is less than the costmin.
- **ValueError** – If the amount_type is volume and the volume is less than the ordermin.
- **ValueError** – If no valid amount_type was passed.

Returns
A string representation of the amount.

Return type
str

Listing 46: Spot Trade: Truncate

```
>>> print(trade.truncate(
...     amount=0.123456789,
...     amount_type="volume",
...     pair="XBTUSD"
... ))
0.12345678

>>> print(trade.truncate(
...     amount=21123.12849829993,
...     amount_type="price",
...     pair="XBTUSD")
... ))
21123.1

>>> print(trade.truncate(
...     amount=0.1,
...     amount_type="volume",
...     pair="XBTUSD"
... ))
0.10000000

>>> print(trade.truncate(
...     amount=21123,
...     amount_type="price",
...     pair="XBTUSD"
... ))
21123.0
```

class kraken.spot.**Funding**(*key: str = ''*, *secret: str = ''*, *url: str = ''*, *proxy: str | None = None*)

Bases: *SpotClient*

Class that implements the Spot Funding client. Currently there are no funding endpoints that could be accesses without authentication.

Parameters

- **key** (*str, optional*) – Spot API public key (default: "")
- **secret** (*str, optional*) – Spot API secret key (default: "")
- **url** (*str, optional*) – Alternative URL to access the Kraken API (default: https://api.kraken.com)
- **proxy** (*str, optional*) – proxy URL, may contain authentication information

Listing 47: Spot Funding: Create the funding client

```
>>> from kraken.spot import Funding
>>> funding = Funding() # unauthenticated
>>> auth_funding = Funding(key="api-key", secret="secret-key") # authenticated
```

Listing 48: Spot Funding: Create the funding client as context manager

```
>>> from kraken.spot import Funding
>>> with Funding(key="api-key", secret="secret-key") as funding:
...     print(funding.get_deposit_methods(asset="XLM"))
```

cancel_withdraw(*asset: str*, *refid: str*, ***, *extra_params: dict | None = None*) → dict

Cancel a requested withdraw. This will only be successful if the withdraw is not being processed so far.

Requires the `Withdraw funds` API key permissions.

- https://docs.kraken.com/api/docs/rest-api/cancel-withdrawal

Parameters

- **asset** (*str*) – Asset of the pending withdraw
- **refid** (*str*) – The reference ID returned the withdraw was requested

Returns
Success or failure

Return type
dict

Listing 49: Spot Funding: Cancel Withdraw

```
>>> from kraken.spot import Funding
>>> funding = Funding(key="api-key", secret="secret-key")
>>> funding.cancel_withdraw(asset="DOT", refid="I7KGS6-UFMTTQ-AGBSO6T")
{ 'result': True }
```

get_deposit_address(*asset: str*, *method: str*, ***, *new: bool | None = False*, *extra_params: dict | None = None*) → list[dict]

Get the deposit addresses for a specific asset. New deposit addresses can be generated.

Requires the `Deposit funds` API key permission.

- https://docs.kraken.com/api/docs/rest-api/get-deposit-addresses

Parameters

- **asset** (*str*) – Asset being deposited
- **method** (*str*) – Deposit method name
- **new** (*bool, optional*) – Generate a new deposit address (default: `False`)

Returns
The user and asset specific deposit addresses

Return type
list[dict]

Listing 50: Spot Funding: Get the available deposit addresses

```
>>> from kraken.spot import Funding
>>> funding = Funding(key="api-key", secret="secret-key")
>>> funding.get_deposit_address(asset="XLM", method="Stellar XLM")
```

(continues on next page)

(continued from previous page)

```
[
    {
        'address': 'GA5XIGA5C7QTPTWXQHY6MCJRMTRZDOSHR6EFIBNDQTCQHG262N4GGKTM',
        'expiretm': '0',
        'new': True,
        'tag': '1668814718654064928'
    }, {
        'address': 'GA5XIGA5C7QTPTWXQHY6MCJRMTRZDOSHR6EFIBNDQTCQHG262N4GGKTM',
        'expiretm': '0',
        'new': True,
        'tag': '1668815609618044006'
    },  ...
]
```

get_deposit_methods(*asset: str*, *, *extra_params: dict | None = None*) → list[dict]

Get the available deposit methods for a specific asset.

- https://docs.kraken.com/api/docs/rest-api/get-deposit-methods

Parameters

asset (*str*) – Asset being deposited

Returns

List of available deposit methods of the asset

Return type

list[dict]

Listing 51: Spot Funding: Get the available deposit methods

```
>>> from kraken.spot import Funding
>>> funding = Funding(key="api-key", secret="secret-key")
>>> funding.get_deposit_methods(asset="XLM")
[
    {
        'method': 'Stellar XLM',
        'limit': False,
        'gen-address': True
    }, {
        'method': 'Stellar XLM Multi',
        'limit': False,
        'gen-address': True
    }
]
```

get_nonce() → str

Return a new nonce

get_recent_deposits_status(*asset: str | None = None*, *method: str | None = None*, *start: str | None = None*, *end: str | None = None*, *cursor: bool | str = False*, *, *extra_params: dict | None = None*) → list[dict] | dict

Get information about the recent deposit status. The look back period is 90 days and only the last 25 deposits will be returned.

Requires the `Query funds` and `Deposit funds` API key permissions.

- https://docs.kraken.com/api/docs/rest-api/get-status-recent-deposits

Parameters

- **asset** (*str, optional*) – Filter by asset
- **method** (*str, optional*) – Filter by deposit method
- **start** (*str, optional*) – Start timestamp
- **end** (*str, optional*) – End timestamp
- **cursor** (bool | str, default: `False`) – If bool: dis-/enable paginated responses; if str: cursor for next page

Returns
The user specific deposit history

Return type
list[dict] | dict

Listing 52: Spot Funding: Get the recent deposit status

```
>>> from kraken.spot import Funding
>>> funding = Funding(key="api-key", secret="secret-key")
>>> funding.get_recent_deposits_status()
[
    {
        'method': 'Bank Frick (SEPA)',
        'aclass': 'currency',
        'asset': 'ZEUR',
        'refid': 'QDFN2EK-XMFWQ4-GPB7SG',
        'txid': '7702226',
        'info': 'NTSBDEB1XXX',
        'amount': '1000.0000',
        'fee': '0.0000',
        'time': 1680245166,
        'status': 'Success'
    }, {
        'method': 'Bank Frick (SEPA)',
        'aclass': 'currency',
        'asset': 'ZEUR',
        'refid': 'QDFO35O-O4IU7K-ZEP77Y',
        'txid': '7559797',
        'info': 'NTSBDEB1XXX',
        'amount': '500.0000',
        'fee': '0.0000',
        'time': 1677827980,
        'status': 'Success'
    }, {
        'method': 'Ethereum (ERC20)',
        'aclass': 'currency',
        'asset': 'XETH',
        'refid': 'Q5QTWMS-GXER37-ZI4IH6',
        'txid':
```

(continues on next page)

(continued from previous page)

```
↪'0x2abb04dafd3caed53d4f2912651391c53b912cc4bca1f8b30d09a5cebec5c2d6',
        'info': '0x35be16f2340c97c02c0cf4dcd9279f2eaa4a0980',
        'amount': '0.0119328120',
        'fee': '0.0000000000',
        'time': 1672901534,
        'status': 'Success'
    }, ...
]
```

get_recent_withdraw_status(*asset: str | None = None*, *method: str | None = None*, *start: str | None = None*, *end: str | None = None*, *cursor: str | bool | None = None*, *, *extra_params: dict | None = None*) → list[dict]

Get information about the recent withdraw status, including withdraws of the past 90 days but at max 500 results.

- https://docs.kraken.com/api/docs/rest-api/get-status-recent-withdrawals

Parameters

- **asset** (*str, optional*) – Filter withdraws by asset (default: None)
- **method** (*str, optional*) – Filter by withdraw method (default: None)
- **start** (*str, optional*) – Filter by start timestamp
- **end** (*str, optional*) – Filter by end timestamp
- **cursor** (*str | bool, optional*) – en-/disable paginated responses via True/False or define the page as str.

Returns
Withdrawal information

Return type
list[dict]

Listing 53: Get the recent withdraw status

```
>>> from kraken.spot import Funding
>>> funding = Funding(key="api-key", secret="secret-key")
>>> funding.get_recent_withdraw_status()
[
    {
        'method': 'Polkadot',
        'aclass': 'currency',
        'asset': 'DOT',
        'refid': 'XXXXXX-XXXXXX-HLDRM5',
        'txid':
↪'0x51d9d13ade1c31a138dae81b845f091d1a6cf2e3c1c36d9cf4f7baf905c483e4',
        'info': '16LrqRXyhjBCSfA6kKrdqxPKrZoMEUtmoW4nkx5ZhA374Bp3',
        'amount': '94.39581164',
        'fee': '0.05000000',
        'time': 1677816733,
        'status': 'Success'
    }, ...
]
```

get_withdrawal_info(*asset: str*, *key: str*, *amount: str | float*, ***, *extra_params: dict | None = None*) → dict

Get information about a possible withdraw, including fee and limit information. The `key` must be the name of the key defined in the account.

Requires the `Query funds` and `Withdraw funds` API key permissions.

- https://docs.kraken.com/api/docs/rest-api/get-withdrawal-information

Parameters

- **asset** (*str*) – Asset to withdraw
- **key** (*str*) – Withdrawal key name as set up in the user account
- **amount** (*str | float*) – The amount to withdraw

Returns
Information about a possible withdraw including the fee and amount.

Return type
dict

Listing 54: Spot Funding: Get withdrawal information

```
>>> from kraken.spot import Funding
>>> funding = Funding(key="api-key", secret="secret-key")
>>> funding.get_withdrawal_info(
...     asset="DOT",
...     key="MyPolkadotWallet",
...     amount="4"
... )
{
    'method': 'Polkadot',
    'limit': '4.49880000',
    'amount': '3.95000000',
    'fee': '0.05000000'
}
```

request(*method: str*, *uri: str*, *params: dict | None = None*, *timeout: int = 10*, ***, *auth: bool = True*, *do_json: bool = False*, *return_raw: bool = False*, *query_str: str | None = None*, *extra_params: str | dict | None = None*) → dict[str, Any] | list[str] | list[dict[str, Any]] | Response

Handles the requested requests, by sending the request, handling the response, and returning the message or in case of an error the respective Exception.

Parameters

- **method** (*str*) – The request method, e.g., `GET`, `POST`, `PUT`, ...
- **uri** (*str*) – The endpoint to send the message
- **auth** (*bool*) – If the requests needs authentication (default: `True`)
- **params** (*dict, optional*) – The query or post parameter of the request (default: `None`)
- **extra_params** (*str | dict, optional*) – Additional query or post parameter of the request (default: `None`)
- **timeout** (*int*) – Timeout for the request (default: `10`)
- **do_json** (*bool*) – If the `params` must be "jsonified" - in case of nested dict style

- **return_raw** (*bool, optional*) – If the response should be returned without parsing. This is used for example when requesting an export of the trade history as .zip archive.
- **query_str** (*str, optional*) – Add custom values to the query /0/public/Nfts?filter%5Bcollection_id%5D=NCQNABO-XYCA7-JMMSDF&page_size=10

Raises
kraken.exceptions.KrakenException.* – If the response contains errors

Returns
The response

Return type
dict | list | requests.Response

property return_unique_id: str

Returns a unique uuid string

Returns
uuid

Return type
str

wallet_transfer(*asset: str*, *from_: str*, *to_: str*, *amount: str | float*, ***, *extra_params: dict | None = None*) → dict

Transfer assets between the Spot and Futures wallet.

Requires the `Withdraw funds` API key permissions.

- https://docs.kraken.com/api/docs/rest-api/wallet-transfer

Parameters

- **asset** (*str*) – Asset to transfer
- **from** (*str*) – The wallet to withdraw from
- **to** (*str*) – The wallet to deposit to
- **amount** (*str | float*) – The amount to transfer

Returns
The reference id

Return type
dict

Listing 55: Spot Funding: Wallet Transfer

```
>>> from kraken.spot import Funding
>>> funding = Funding(key="api-key", secret="secret-key")
>>> funding.wallet_transfer(
...     asset="XBT",
...     from_="Spot Wallet",
...     to_="Futures Wallet",
...     amount=0.01
... )
{ 'refid': "ANS1EE5-SKACR4-PENGVP" }
```

withdraw_addresses(*asset: str | None = None*, *aclass: str | None = None*, *method: str | None = None*, *key: str | None = None*, *verified: bool | None = None*, ***, *extra_params: dict | None = None*) → dict

Returns the list of available withdrawal addresses for that user.

Requires the `Funds permissions - Query` and `Funds permissions - Withdraw` API key permissions.

Parameters

- **asset** (*Optional[str]*) – Filter by asset
- **aclass** (*Optional[str]*) – Filter by asset class (default: `currency`)
- **method** (*Optional[str]*) – Filter by method
- **key** (*Optional[str]*) – Filter by key
- **verified** (*Optional[str]*) – List only addresses which are confirmed via E-Mail

Returns
List of available addresses for withdrawal

Return type
list[dict]

withdraw_funds(*asset: str*, *key: str*, *amount: str | float*, *max_fee: str | None = None*, ***, *extra_params: dict | None = None*) → dict

Create a new withdraw. The key must be the name of the withdraw key defined in the withdraw section of the Kraken WebUI.

Requires the `Withdraw funds` API key permissions.

- https://docs.kraken.com/api/docs/rest-api/withdraw-funds

Parameters

- **asset** (*str*) – Asset to withdraw
- **key** (*str*) – Withdrawal key name as set up in the account
- **amount** (*str | float*) – The amount to withdraw
- **max_fee** (*str*) – Fail withdraw if the fee will be higher than the specified max_fee.

Returns
The reference id of the withdraw

Return type
dict

Listing 56: Spot Funding: Withdraw Funds

```
>>> from kraken.spot import Funding
>>> funding = Funding(key="api-key", secret="secret-key")
>>> funding.withdraw_funds(
...     asset="DOT",
...     key="MyPolkadotWallet"
...     amount=4
... )
{'refid': 'I7KGS6-UFMTTQ-AGBSO6T'}
```

withdraw_methods(*asset: str | None = None*, *aclass: str | None = None*, *network: str | None = None*, ***, *extra_params: dict | None = None*) → dict

Returns the list of available withdraw methods for that user.

Requires the `Funds permissions - Query` and `Funds permissions - Withdraw` API key permissions.

Parameters

- **asset** (*Optional[str]*) – Filter by asset
- **aclass** (*Optional[str]*) – Filter by asset class (default: `currency`)
- **network** (*Optional[str]*) – Filter by network

Returns
List of available withdraw methods

Return type
list[dict]

class kraken.spot.**Earn**(*key: str = ''*, *secret: str = ''*, *url: str = ''*, *proxy: str | None = None*)

Bases: *SpotClient*

Class that implements the Kraken Spot Earn client. Currently there are no earn endpoints that could be accesses without authentication.

Parameters

- **key** (*str, optional*) – Spot API public key (default: "")
- **secret** (*str, optional*) – Spot API secret key (default: "")
- **url** (*str, optional*) – Alternative URL to access the Kraken API (default: https://api.kraken.com)
- **proxy** (*str, optional*) – proxy URL, may contain authentication information

Listing 57: Spot Earn: Create the Earn client

```
>>> from kraken.spot import Earn
>>> earn = Earn() # unauthenticated
>>> auth_earn = Earn(key="api-key", secret="secret-key") # authenticated
```

Listing 58: Spot Earn: Create the earn client as context manager

```
>>> from kraken.spot import Earn
>>> with Earn(key="api-key", secret="secret-key") as earn:
...     print(earn.stake_asset(asset="XLM", amount=200, method="Lumen Staked"))
```

allocate_earn_funds(*amount: str | float*, *strategy_id: str*, ***, *extra_params: dict | None = None*) → bool

Allocate funds according to the defined strategy.

Requires the `Earn  Funds` API key permission

- https://docs.kraken.com/api/docs/rest-api/allocate-strategy

Parameters

- **amount** (*str | float*) – The amount to allocate

- **strategy_id** (*str*) – Identifier of th chosen earn strategy (see *kraken.spot.Earn.list_earn_strategies()*)

Listing 59: Spot Earn: Allocate funds

```
>>> from kraken.earn import Earn
>>> earn = Earn(key="api-key", secret="secret-key")
>>> earn.allocate_earn_funds(
...     amount=2000,
...     strategy_id="ESRFUO3-Q62XD-WIOIL7"
... )
True
```

deallocate_earn_funds(*amount: str | float*, *strategy_id: str*, ***, *extra_params: dict | None = None*) → bool

Deallocate funds according to the defined strategy.

Requires the `Earn Funds` API key permission

- https://docs.kraken.com/api/docs/rest-api/deallocate-strategy

Parameters

- **amount** (*str | float*) – The amount to deallocate
- **strategy_id** (*str*) – Identifier of th chosen earn strategy (see *kraken.spot.Earn.list_earn_strategies()*)

Listing 60: Spot Earn: Deallocate funds

```
>>> from kraken.earn import Earn
>>> earn = Earn(key="api-key", secret="secret-key")
>>> earn.deallocate_earn_funds(
...     amount=2000,
...     strategy_id="ESRFUO3-Q62XD-WIOIL7"
... )
True
```

get_allocation_status(*strategy_id: str*, ***, *extra_params: dict | None = None*) → dict

Retrieve the status of the last allocation request.

Requires the `Earn Funds` or `Query Funds` API key permission.

- https://docs.kraken.com/api/docs/rest-api/get-allocate-strategy-status

Parameters

strategy_id (*str*) – Identifier of th chosen earn strategy (see *kraken.spot.Earn.list_earn_strategies()*)

Listing 61: Spot Earn: Allocation Status

```
>>> from kraken.earn import Earn
>>> earn = Earn(key="api-key", secret="secret-key")
>>> earn.get_allocation_status(
...     strategy_id="ESRFUO3-Q62XD-WIOIL7"
... )
{'pending': False}
```

get_deallocation_status(*strategy_id: str*, *, *extra_params: dict | None = None*) → dict

Retrieve the status of the last deallocation request.

Requires the `Earn Funds` or `Query Funds` API key permission.

- https://docs.kraken.com/api/docs/rest-api/get-deallocate-strategy-status

Parameters

strategy_id (*str*) – Identifier of th chosen earn strategy (see *kraken.spot.Earn.list_earn_strategies()*)

Listing 62: Spot Earn: Deallocation Status

```
>>> from kraken.earn import Earn
>>> earn = Earn(key="api-key", secret="secret-key")
>>> earn.get_deallocation_status(
...     strategy_id="ESRFUO3-Q62XD-WIOIL7"
... )
{'pending': False}
```

get_nonce() → str

Return a new nonce

list_earn_allocations(*ascending: str | None = None*, *hide_zero_allocations: str | None = None*, *converted_asset: str | None = None*, *, *extra_params: dict | None = None*) → dict

List the user's allocations.

Requires the `Query Funds` API key permission.

- https://docs.kraken.com/api/docs/rest-api/list-allocations

(March 9, 2024): The endpoint is not fully implemented on the side of Kraken. Some errors may happen.

Parameters

- **ascending** (*bool, optional*) – Sort ascending, defaults to False
- **hide_zero_allocations** (*bool, optional*) – Hide past allocations without balance, defaults to False
- **coverted_asset** (*str, optional*) – Currency to express the value of the allocated asset, defaults to None

Listing 63: Spot Earn: List Earn Allocations

```
>>> from kraken.earn import Earn
>>> earn = Earn(key="api-key", secret="secret-key")
>>> earn.list_earn_allocations(asset="DOT")
{
"converted_asset": "USD",
    "total_allocated": "49.2398",
    "total_rewarded": "0.0675",
    "next_cursor": "2",
    "items": [{
        "strategy_id": "ESDQCOL-WTZEU-NU55QF",
        "native_asset": "ETH",
        "amount_allocated": {},
        "total_rewarded": {}
```

(continues on next page)

(continued from previous page)

```
    }]
}
```

list_earn_strategies(*asset: str | None = None*, *limit: int | None = None*, *lock_type: list[str] | None = None*, *cursor: bool | None = None*, *ascending: bool | None = None*, *, extra_params: dict | None = None*) → dict

List the available earn strategies as well as additional information.

Requires an API key but no special permission set.

- https://docs.kraken.com/api/docs/rest-api/list-strategies

(March 9, 2024): The endpoint is not fully implemented on the side of Kraken. Some errors may happen.

Parameters

- **asset** (*Optional[str], optional*) – Asset to filter for, defaults to None
- **limit** (*Optional[int], optional*) – Items per page, defaults to None
- **lock_type** (*Optional[list[str]], optional*) – Filter strategies by lock type (`flex`, bounded, `timed`, `instant`), defaults to None
- **cursor** (*Optional[bool], optional*) – Page ID, defaults to None
- **ascending** (*bool, optional*) – Sort ascending, defaults to False

Listing 64: Spot Earn: List Earn Strategies

```
>>> from kraken.earn import Earn
>>> earn = Earn(key="api-key", secret="secret-key")
>>> earn.list_earn_strategies(asset="DOT")
{
    "next_cursor": None,
    "items": [
        {
            "id": "ESMWVX6-JAPVY-23L3CV",
            "asset": "DOT",
            "lock_type": {
                "type": "bonded",
                "payout_frequency": 604800,
                "bonding_period": 0,
                "bonding_period_variable": False,
                "bonding_rewards": False,
                "unbonding_period": 2419200,
                "unbonding_period_variable": False,
                "unbonding_rewards": False,
                "exit_queue_period": 0,
            },
            "apr_estimate": {"low": "15.0000", "high": "21.0000"},
            "user_min_allocation": "0.01",
            "allocation_fee": "0.0000",
            "deallocation_fee": "0.0000",
            "auto_compound": {"type": "enabled"},
            "yield_source": {"type": "staking"},
            "can_allocate": True,
```

(continues on next page)

(continued from previous page)

```
            "can_deallocate": True,
            "allocation_restriction_info": [],
        },
        {
            "id": "ESRFUO3-Q62XD-WIOIL7",
            "asset": "DOT",
            "lock_type": {"type": "instant", "payout_frequency": 604800},
            "apr_estimate": {"low": "7.0000", "high": "11.0000"},
            "user_min_allocation": "0.01",
            "allocation_fee": "0.0000",
            "deallocation_fee": "0.0000",
            "auto_compound": {"type": "enabled"},
            "yield_source": {"type": "staking"},
            "can_allocate": True,
            "can_deallocate": True,
            "allocation_restriction_info": [],
        },
    ],
}
```

request(*method: str*, *uri: str*, *params: dict | None = None*, *timeout: int = 10*, ***, *auth: bool = True*, *do_json: bool = False*, *return_raw: bool = False*, *query_str: str | None = None*, *extra_params: str | dict | None = None*) → dict[str, Any] | list[str] | list[dict[str, Any]] | Response

Handles the requested requests, by sending the request, handling the response, and returning the message or in case of an error the respective Exception.

Parameters

- **method** (*str*) – The request method, e.g., `GET`, `POST`, `PUT`, ...
- **uri** (*str*) – The endpoint to send the message
- **auth** (*bool*) – If the requests needs authentication (default: `True`)
- **params** (*dict, optional*) – The query or post parameter of the request (default: `None`)
- **extra_params** (*str | dict, optional*) – Additional query or post parameter of the request (default: `None`)
- **timeout** (*int*) – Timeout for the request (default: `10`)
- **do_json** (*bool*) – If the `params` must be "jsonified" - in case of nested dict style
- **return_raw** (*bool, optional*) – If the response should be returned without parsing. This is used for example when requesting an export of the trade history as .zip archive.
- **query_str** (*str, optional*) – Add custom values to the query /0/public/Nfts?filter%5Bcollection_id%5D=NCQNABO-XYCA7-JMMSDF&page_size=10

Raises

kraken.exceptions.KrakenException.* – If the response contains errors

Returns

The response

Return type

dict | list | requests.Response

property return_unique_id: str

Returns a unique uuid string

Returns
uuid

Return type
str

CHAPTER

SEVEN

SPOT WEBSOCKETS

class kraken.spot.**SpotWSClient**(*key: str = ''*, *secret: str = ''*, *callback: Callable | None = None*, ***, *no_public: bool = False*, *rest_url: str | None = None*, *ws_url: str | None = None*, *auth_ws_url: str | None = None*)

Bases: `SpotWSClientBase`

This client only supports the Kraken Websocket API v2.

Class to access public and private/authenticated websocket connections.

- https://docs.kraken.com/websockets-v2

This class holds up to two websocket connections, one private and one public. The core functionalities are un-/subscribing to websocket feeds and sending messages. See *kraken.spot.SpotWSClient.subscribe()* and `kraken.spot.SpotWSClientV.send_message()` for more information.

When accessing private endpoints that need authentication make sure, that the `Access WebSockets API` API key permission is set in the user's account. To place or cancel orders, querying ledger information or accessing live portfolio changes (fills, new orders, ...) there are separate permissions that must be enabled if required.

Parameters

- **key** (*str, optional*) – API Key for the Kraken Spot API (default: `""`)
- **secret** (*str, optional*) – Secret API Key for the Kraken Spot API (default: `""`)
- **url** (*str, optional*) – Set a specific URL to access the Kraken REST API
- **no_public** – Disables public connection (default: `False`). If not set or set to `False`, the client will create a public and a private connection per default. If only a private connection is required, this parameter should be set to `True`.
- **rest_url** (*str, optional*) – Set a specific REST URL to access the Kraken Spot API (default: `None`).
- **ws_url** (*str, optional*) – Set a specific Websocket URL to access the Kraken Spot API (default: `None`).
- **auth_ws_url** – Set a specific Authenticated Websocket URL to access the Kraken Spot API (default: `None`).

Listing 1: HowTo: Use the Kraken Spot websocket client

```
import asyncio
from kraken.spot import SpotWSClient


class Client(SpotWSClient):
```

(continues on next page)

(continued from previous page)

```

    async def on_message(self, message):
        print(message)


async def main():
    client = Client()         # unauthenticated
    client_auth = Client(     # authenticated
        key="kraken-api-key",
        secret="kraken-secret-key"
    )
    # open the websocket connections
    await client.start()
    await auth_client.start()

    # subscribe to the desired feeds:
    await client.subscribe(
        params={"channel": "ticker", "symbol": ["BTC/USD"]}
    )
    # from now on the on_message function receives the ticker feed

    while not client.exception_occur:
        await asyncio.sleep(6)

if __name__ == "__main__":
    try:
        asyncio.run(main())
    except KeyboardInterrupt:
        pass
```

Listing 2: HowTo: Use the websocket client as instance

```
import asyncio
from kraken.spot import SpotWSClient


async def on_message(message):
    print(message)

async def main():

    client = SpotWSClient(callback=on_message)
    await client.start()
    await client.subscribe(
        params={"channel": "ticker", "symbol": ["BTC/USD"]}
    )

    while not client.exception_occur:
        await asyncio.sleep(10)


if __name__ == "__main__":
```

(continues on next page)

(continued from previous page)

```
    try:
        asyncio.run(main())
    except KeyboardInterrupt:
        pass
```

Listing 3: HowTo: Use the websocket client as context manager

```
import asyncio
from kraken.spot import SpotWSClient

async def on_message(message):
    print(message)

async def main():
    async with SpotWSClient(
        key="api-key",
        secret="secret-key",
        callback=on_message
    ) as session:
        await session.subscribe(
            params={"channel": "ticker", "symbol": ["BTC/USD"]}
        )

    while True
        await asyncio.sleep(6)


if __name__ == "__main__":
    try:
        asyncio.run(main())
    except KeyboardInterrupt:
        pass
```

Listing 4: HowTo: How to connect to other Kraken instances

```
client_auth = SpotWSClient(
    key="api-key",
    secret="secret-key",
    rest_url="https://api.vip.uat.lobster.kraken.com",
    ws_url="wss://ws.vip.uat.lobster.kraken.com",
    auth_ws_url="wss://ws-auth.vip.uat.lobster.kraken.com",
)
```

property active_private_subscriptions: list[dict]

Returns the active private subscriptions

Returns
List of active private subscriptions

Return type
list[dict]

Raises
ConnectionError – If there is no active private connection

property active_public_subscriptions: list[dict]

Returns the active public subscriptions

Returns
List of active public subscriptions

Return type
list[dict]

Raises
ConnectionError – If there is no active public connection.

async_close() → None

Closes the aiohttp session

async close() → None

Method to close the websocket connection.

property exception_occur: bool

Returns True if any connection was stopped due to an exception.

get_nonce() → str

Return a new nonce

async get_ws_token() → dict

Get the authentication token to establish the authenticated websocket connection. This is used internally and in most cases not needed outside.

- https://docs.kraken.com/api/docs/guides/spot-ws-auth

Returns
The authentication token

Return type
dict

async on_message(*message: dict | list*) → None

Calls the defined callback function (if defined). In most cases you have to overwrite this function since it will receive all incoming messages that will be sent by Kraken.

See *kraken.spot.SpotWSClient* for examples to use this function.

Parameters
message (*dict | list*) – The message received sent by Kraken via the websocket connection

property private_channel_names: list[str]

Returns the list of valid values for `channel` when un-/subscribing from/to private feeds that need authentication.

Override this property if the exchange supports additional private channels.

- executions
- balances
- level3

Returns
List of available private channel names

Return type
list[str]

property private_methods: list[str]

Returns the list of available methods - parameters are similar to the REST API trade methods.

The available methods and their documentation are listed below (as of June 2023):

- add_order
- *amend_order* <https://docs.kraken.com/api/docs/websocket-v2/amend_order>`_
- cancel_order
- cancel_all
- cancel_all_orders_after
- batch_order
- batch_cancel
- edit_order

Returns
List of available methods

Return type
list[str]

property public_channel_names: list[str]

Returns the list of valid values for `channel` when un-/subscribing from/to public feeds without authentication.

Override this property if the exchange supports additional public channels.

The available public channels are listed below:

- book
- instrument
- ohlc
- ticker
- trade

Returns
List of available public channel names

Return type
list[str]

async request(*method: str*, *uri: str*, *params: dict | None = None*, *timeout: int = 10*, ***, *auth: bool = True*, *do_json: bool = False*, *return_raw: bool = False*, *query_str: str | None = None*, *extra_params: str | dict | None = None*) → Coroutine

Handles the requested requests, by sending the request, handling the response, and returning the message or in case of an error the respective Exception.

Parameters

- **method** (*str*) – The request method, e.g., `GET`, `POST`, `PUT`, ...

- **uri** (*str*) – The endpoint to send the message
- **auth** (*bool*) – If the requests needs authentication (default: True)
- **params** (*dict, optional*) – The query or post parameter of the request (default: None)
- **timeout** (*int*) – Timeout for the request (default: 10)
- **do_json** (*bool*) – If the params must be "jsonified" - in case of nested dict style
- **return_raw** (*bool, optional*) – If the response should be returned without parsing. This is used for example when requesting an export of the trade history as .zip archive.
- **query_str** (*str, optional*) – Add custom values to the query /0/public/Nfts?filter%5Bcollection_id%5D=NCQNABO-XYCA7-JMMSDF&page_size=10

Raises
kraken.exceptions.KrakenException.* – If the response contains errors

Returns
The response

Return type
dict | list | aiohttp.ClientResponse

property return_unique_id: str

Returns a unique uuid string

Returns
uuid

Return type
str

async send_message(*message: dict*, *, *raw: bool = False*) → None

Sends a message via the websocket connection. For private messages the authentication token will be assigned automatically if raw=False.

The user can specify a req_d within the message to identify corresponding responses via websocket feed.

Parameters

- **message** (*dict*) – The information to send
- **raw** (*bool, optional*) – If set to True the message will be sent directly.

The following examples demonstrate how to use the *kraken.spot.SpotWSClient.send_message()* function. The client must be instantiated as described in *kraken.spot.SpotWSClient* where client uses public connections (without authentication) and client_auth must be instantiated using valid credentials since only this way placing or canceling orders can be done.

Please note that the send_message function will automatically pass the authentication token (except for the case if raw=True).

Placing orders using an authenticated websocket connection can be easily done as shown in the example below. See https://docs.kraken.com/api/docs/websocket-v2/add_order to retrieve more information about the available parameters.

Listing 5: Spot Websocket: Place a new order

```
>>> await client_auth.send_message(
...     message={
...         "method": "add_order",
...         "params": {
...             "limit_price": 1234.56,
...             "order_type": "limit",
...             "order_userref": 123456789,
...             "order_qty": 1.0,
...             "side": "buy",
...             "symbol": "BTC/USD",
...         },
...     }
... )
```

Placing orders as batch can be done by passing `batch_add` as method. Its parameters and limitations are described in https://docs.kraken.com/api/docs/websocket-v2/batch_add.

Listing 6: Spot Websocket: Placing orders as batch

```
>>> await client_auth.send_message(
...     message={
...         "method": "batch_add",
...         "params": {
...             "orders": [
...                 {
...                     "limit_price": 1000.23,
...                     "order_qty": 1,
...                     "order_type": "limit",
...                     "order_userref": 123456789,
...                     "side": "buy",
...                 },
...                 {
...                     "limit_price": 500.21,
...                     "order_qty": 2.12345,
...                     "order_type": "limit",
...                     "order_userref": 212345679,
...                     "side": "sell",
...                     "stp_type": "cancel_both",
...                 },
...             ],
...             "symbol": "BTC/USD",
...             "validate": True,
...         },
...     }
... )
```

Cancel orders as batch is available using the `batch_cancel` method as described in https://docs.kraken.com/api/docs/websocket-v2/batch_cancel.

Listing 7: Spot Websocket: Cancel orders as batch

```
>>> await client_auth.send_message(
...     message={
...         "method": "batch_cancel",
...         "params": {
...             "orders": [
...                 "123456789",
...                 "212345679",
...                 "ORDER-ID123-4567890"
...             ],
...         },
...     }
... )
```

Cancel all orders can be used as the name suggests - to cancel all open orders (see https://docs.kraken.com/api/docs/websocket-v2/cancel_all).

Listing 8: Spot Websocket: Cancel all orders

```
>>> await client_auth.send_message(
...     message={
...         "method": "cancel_all",
...     }
... )
```

Death Man's Switch is a useful utility to reduce the risk of losses due to network fuck-ups since it will cancel all orders if the call was not received by Kraken within a certain amount of time. See https://docs.kraken.com/api/docs/websocket-v2/cancel_after for more information.

Listing 9: Spot Websocket: Death Man's Switch / cancel_all_orders_after

```
>>> await client_auth.send_message(
...     message={
...         "method": "cancel_all_orders_after",
...         "params": {"timeout": 60},
...     }
... )
```

Canceling orders is a common task during trading and can be done as described in https://docs.kraken.com/api/docs/websocket-v2/cancel_order.

Listing 10: Spot Websocket: Cancel order(s)

```
>>> await client_auth.send_message(
...     message={
...         "method": "cancel_order",
...         "params": {
...             "order_id": ["ORDER-ID123-456789", "ORDER-ID123-987654"],
...         },
...     }
... )
```

Editing orders can be done as shown in the example below. See https://docs.kraken.com/api/docs/websocket-v2/edit_order for more information.

Listing 11: Spot Websocket: Cancel order(s)

```
>>> await client_auth.send_message(
...     message={
...         "method": "edit_order",
...         "params": {
...             "order_id": "ORDER-ID123-456789",
...             "order_qty": 2.5,
...             "symbol": "BTC/USD",
...         },
...     }
... )
```

Subscribing to websocket feeds can be done using the send_message function but it is recommended to use *kraken.spot.SpotWSClient.subscribe()* instead.

Listing 12: Spot Websocket: Subscribe to a websocket feed

```
>>> await client.send_message(
...     message={
...         "method": "subscribe",
...         "params": {"channel": "book", "snapshot": False, "symbol": ["BTC/USD
↪"]},
...     }
... )
```

async start() → None

Method to start the websocket connection.

stop() → None

Method to stop the websocket connection.

async subscribe(*params: dict*, *req_id: int | None = None*) → None

Subscribe to a channel/feed

Success or failures are sent over the websocket connection and can be received via the on_message or callback function.

When accessing private endpoints and subscription feeds that need authentication make sure that the `Access WebSockets API` API key permission is set in the users Kraken account.

Please note that this function automatically assigns the `method` key and sets its value to `subscribe`. The authentication token is also assigned automatically, so only the `params` are needed here.

Parameters

- **params** (*dict*) – The subscription message
- **req_id** (*int, optional*) – Identification number that will be added to the response message sent by the websocket feed.

Initialize your client as described in *kraken.spot.SpotWSClient* to run the following example:

Listing 13: Spot Websocket: Subscribe to a websocket feed

```
>>> await client.subscribe(
...     params={"channel": "ticker", "symbol": ["BTC/USD"]}
... )
```

async unsubscribe(*params: dict*, *req_id: int | None = None*) → None

Unsubscribe from a channel/feed

Success or failures are sent via the websocket connection and can be received via the on_message or callback function.

When accessing private endpoints and subscription feeds that need authentication make sure, that the `Access WebSockets API` API key permission is set in the users Kraken account.

Parameters

params (*dict*) – The un-subscription message (only the params part)

Initialize your client as described in *kraken.spot.SpotWSClient* to run the following example:

Listing 14: Spot Websocket: Unsubscribe from a websocket feed

```
>>> await client.unsubscribe(
...     params={"channel": "ticker", "symbol": ["BTC/USD"]}
... )
```

class kraken.spot.**SpotOrderBookClient**(*depth: int = 10*, *callback: Callable | None = None*)

Bases: *SpotWSClient*

This client is using the Kraken Websocket API v2

The orderbook client can be used for instantiation and maintaining one or multiple order books for Spot trading on the Kraken Crypto Asset Exchange. It uses websockets to subscribe to book feeds and receives book updates, calculates the checksum and will publish the raw message to the *on_book_update()* function or to the specified callback function.

get() can be used to access a specific book of this client - they will always be up-to date when used from within *on_book_update()*.

The client will resubscribe to the book feed(s) if any errors occur and publish the changes to the mentioned function(s). This is required to compute the correct checksum internally.

This class has a default book depth of 10. Available depths are: 10, 25, 50, 100, 500, 1000. This client can handle multiple books - but only for one depth. When subscribing to books with different depths, please use separate instances of this class.

- https://docs.kraken.com/api/docs/guides/spot-ws-book-v1

Listing 15: Example: Create and maintain a Spot orderbook as custom class

```python
from typing import Any
from kraken.spot import SpotOrderBookClient
import asyncio

class OrderBook(SpotOrderBookClient):
    async def on_book_update(self: "OrderBook", pair: str, message:
    list) -> None:
        '''This function must be overloaded to get the recent updates.'''
        book: dict[str, Any] = self.get(pair=pair) bid:s
        list[tuple[str, str]] = list(book["bid"].items()) ask:
        list[tuple[str, str]] = list(book["ask"].items())

        print("Bid         Volume                Ask         Volume") for level in
        range(self.depth):
            print(
                f"{bid[level][0]} ({bid[level][1]})      {ask[level][0]}
                ({ask[level][1]})"
            )

async def main() -> None:
    orderbook: OrderBook = OrderBook(depth=10)
    await orderbook.start()

    await orderbook.add_book(
        pairs=["XBT/USD"]  # we can also subscribe to more currency
        pairs
    )

    while not orderbook.exception_occur:
        await asyncio.sleep(10)

if __name__ == "__main__":
    try:
        asyncio.run(main())
    except KeyboardInterrupt:
        pass
```

Listing 16: Example: Create and maintain a Spot orderbook using a callback

```python
from typing import Any
from kraken.spot import SpotOrderBookClient
import asyncio

async def my_callback(self: "OrderBook", pair: str, message: dict) -> None:
    '''This function do not need to be async.'''
    print(message)

async def main() -> None:
```

(continues on next page)

(continued from previous page)

```
    orderbook: OrderBook = OrderBook(depth=100, callback=my_callback)
    await orderbook.start()

    await orderbook.add_book(
        pairs=["XBT/USD"]  # we can also subscribe to more currency
        pairs
    )

    while not orderbook.exception_occur:
        await asyncio.sleep(10)

if __name__ == "__main__":
    try:
        asyncio.run(main())
    except KeyboardInterrupt:
        pass
```

property active_private_subscriptions: list[dict]

Returns the active private subscriptions

Returns
List of active private subscriptions

Return type
list[dict]

Raises
ConnectionError – If there is no active private connection

property active_public_subscriptions: list[dict]

Returns the active public subscriptions

Returns
List of active public subscriptions

Return type
list[dict]

Raises
ConnectionError – If there is no active public connection.

async add_book(*pairs: list[str]*) → None

Add an orderbook to this client. The feed will be subscribed and updates will be published to the *on_book_update()* function.

Parameters

- **pairs** (*list[str]*) – The pair(s) to subscribe to
- **depth** (*int*) – The book depth

async_close() → None

Closes the aiohttp session

async close() → None

Method to close the websocket connection.

property depth: int

Return the fixed depth of this orderbook client.

property exception_occur: bool

Returns True if any connection was stopped due to an exception.

get(*pair: str*) → dict | None

Returns the orderbook for a specific `pair`.

Parameters
pair (*str*) – The pair to get the orderbook from

Returns
The orderbook of that `pair`.

Return type
dict

static get_first(*values: tuple*) → float

This function is used as callback for the `sorted` method to sort a tuple/list by its first value and while ensuring that the values are floats and comparable.

Parameters
values (*tuple*) – A tuple of string values

Returns
The first value of `values` as float.

Return type
float

get_nonce() → str

Return a new nonce

async get_ws_token() → dict

Get the authentication token to establish the authenticated websocket connection. This is used internally and in most cases not needed outside.

- https://docs.kraken.com/api/docs/guides/spot-ws-auth

Returns
The authentication token

Return type
dict

async on_book_update(*pair: str*, *message: dict*) → None

This function will be called every time the orderbook gets updated. It needs to be overloaded if no callback function was defined during the instantiation of this class.

Parameters

- **pair** (*str*) – The currency pair of the orderbook that has been updated.
- **message** (*dict*) – The book message sent by Kraken

async on_message(*message: list | dict*) → None

This function must not be overloaded - it would break this client!

It receives and processes the book related websocket messages and is only publicly visible for those who understand and are willing to mock it.

property private_channel_names: list[str]

Returns the list of valid values for `channel` when un-/subscribing from/to private feeds that need authentication.

Override this property if the exchange supports additional private channels.

- executions
- balances
- level3

Returns
List of available private channel names

Return type
list[str]

property private_methods: list[str]

Returns the list of available methods - parameters are similar to the REST API trade methods.

The available methods and their documentation are listed below (as of June 2023):

- add_order
- *amend_order* <https://docs.kraken.com/api/docs/websocket-v2/amend_order>`_
- cancel_order
- cancel_all
- cancel_all_orders_after
- batch_order
- batch_cancel
- edit_order

Returns
List of available methods

Return type
list[str]

property public_channel_names: list[str]

Returns the list of valid values for `channel` when un-/subscribing from/to public feeds without authentication.

Override this property if the exchange supports additional public channels.

The available public channels are listed below:

- book
- instrument
- ohlc
- ticker
- trade

Returns
List of available public channel names

Return type
list[str]

`async remove_book(pairs: list[str]) → None`

Unsubscribe from a subscribed orderbook.

Parameters

- **pairs** (*list[str]*) – The pair(s) to unsubscribe from
- **depth** (*int*) – The book depth

`async request(method: str, uri: str, params: dict | None = None, timeout: int = 10, *, auth: bool = True, do_json: bool = False, return_raw: bool = False, query_str: str | None = None, extra_params: str | dict | None = None) → Coroutine`

Handles the requested requests, by sending the request, handling the response, and returning the message or in case of an error the respective Exception.

Parameters

- **method** (*str*) – The request method, e.g., `GET`, `POST`, `PUT`, ...
- **uri** (*str*) – The endpoint to send the message
- **auth** (*bool*) – If the requests needs authentication (default: `True`)
- **params** (*dict, optional*) – The query or post parameter of the request (default: `None`)
- **timeout** (*int*) – Timeout for the request (default: `10`)
- **do_json** (*bool*) – If the `params` must be "jsonified" - in case of nested dict style
- **return_raw** (*bool, optional*) – If the response should be returned without parsing. This is used for example when requesting an export of the trade history as .zip archive.
- **query_str** (*str, optional*) – Add custom values to the query /0/public/Nfts?filter%5Bcollection_id%5D=NCQNABO-XYCA7-JMMSDF&page_size=10

Raises
kraken.exceptions.KrakenException.* – If the response contains errors

Returns
The response

Return type
dict | list | aiohttp.ClientResponse

`property return_unique_id: str`

Returns a unique uuid string

Returns
uuid

Return type
str

async send_message(*message: dict*, *, *raw: bool = False*) → None

Sends a message via the websocket connection. For private messages the authentication token will be assigned automatically if `raw=False`.

The user can specify a `req_d` within the message to identify corresponding responses via websocket feed.

Parameters

- **message** (*dict*) – The information to send
- **raw** (*bool, optional*) – If set to `True` the `message` will be sent directly.

The following examples demonstrate how to use the *kraken.spot.SpotWSClient.send_message()* function. The client must be instantiated as described in *kraken.spot.SpotWSClient* where `client` uses public connections (without authentication) and `client_auth` must be instantiated using valid credentials since only this way placing or canceling orders can be done.

Please note that the send_message function will automatically pass the authentication token (except for the case if `raw=True`).

Placing orders using an authenticated websocket connection can be easily done as shown in the example below. See https://docs.kraken.com/api/docs/websocket-v2/add_order to retrieve more information about the available parameters.

Listing 17: Spot Websocket: Place a new order

```
>>> await client_auth.send_message(
...     message={
...         "method": "add_order",
...         "params": {
...             "limit_price": 1234.56,
...             "order_type": "limit",
...             "order_userref": 123456789,
...             "order_qty": 1.0,
...             "side": "buy",
...             "symbol": "BTC/USD",
...         },
...     }
... )
```

Placing orders as batch can be done by passing `batch_add` as method. Its parameters and limitations are described in https://docs.kraken.com/api/docs/websocket-v2/batch_add.

Listing 18: Spot Websocket: Placing orders as batch

```
>>> await client_auth.send_message(
...     message={
...         "method": "batch_add",
...         "params": {
...             "orders": [
...                 {
...                     "limit_price": 1000.23,
...                     "order_qty": 1,
...                     "order_type": "limit",
...                     "order_userref": 123456789,
...                     "side": "buy",
...                 },
```

(continues on next page)

(continued from previous page)

```
...                 {
...                     "limit_price": 500.21,
...                     "order_qty": 2.12345,
...                     "order_type": "limit",
...                     "order_userref": 212345679,
...                     "side": "sell",
...                     "stp_type": "cancel_both",
...                 },
...             ],
...             "symbol": "BTC/USD",
...             "validate": True,
...         },
...     }
... )
```

Cancel orders as batch is available using the `batch_cancel` method as described in https://docs.kraken.com/api/docs/websocket-v2/batch_cancel.

Listing 19: Spot Websocket: Cancel orders as batch

```
>>> await client_auth.send_message(
...     message={
...         "method": "batch_cancel",
...         "params": {
...             "orders": [
...                 "123456789",
...                 "212345679",
...                 "ORDER-ID123-4567890"
...             ],
...         },
...     }
... )
```

Cancel all orders can be used as the name suggests - to cancel all open orders (see https://docs.kraken.com/api/docs/websocket-v2/cancel_all).

Listing 20: Spot Websocket: Cancel all orders

```
>>> await client_auth.send_message(
...     message={
...         "method": "cancel_all",
...     }
... )
```

Death Man's Switch is a useful utility to reduce the risk of losses due to network fuck-ups since it will cancel all orders if the call was not received by Kraken within a certain amount of time. See https://docs.kraken.com/api/docs/websocket-v2/cancel_after for more information.

Listing 21: Spot Websocket: Death Man's Switch / cancel_all_orders_after

```
>>> await client_auth.send_message(
...     message={
```

(continues on next page)

(continued from previous page)

```
...         "method": "cancel_all_orders_after",
...         "params": {"timeout": 60},
...     }
... )
```

Canceling orders is a common task during trading and can be done as described in https://docs.kraken.com/api/docs/websocket-v2/cancel_order.

Listing 22: Spot Websocket: Cancel order(s)

```
>>> await client_auth.send_message(
...     message={
...         "method": "cancel_order",
...         "params": {
...             "order_id": ["ORDER-ID123-456789", "ORDER-ID123-987654"],
...         },
...     }
... )
```

Editing orders can be done as shown in the example below. See https://docs.kraken.com/api/docs/websocket-v2/edit_order for more information.

Listing 23: Spot Websocket: Cancel order(s)

```
>>> await client_auth.send_message(
...     message={
...         "method": "edit_order",
...         "params": {
...             "order_id": "ORDER-ID123-456789",
...             "order_qty": 2.5,
...             "symbol": "BTC/USD",
...         },
...     }
... )
```

Subscribing to websocket feeds can be done using the send_message function but it is recommended to use *kraken.spot.SpotWSClient.subscribe()* instead.

Listing 24: Spot Websocket: Subscribe to a websocket feed

```
>>> await client.send_message(
...     message={
...         "method": "subscribe",
...         "params": {"channel": "book", "snapshot": False, "symbol": ["BTC/USD
↪"]},
...     }
... )
```

async start() → None

Method to start the websocket connection.

stop() → None

Method to stop the websocket connection.

async subscribe(*params: dict*, *req_id: int | None = None*) → None

Subscribe to a channel/feed

Success or failures are sent over the websocket connection and can be received via the on_message or callback function.

When accessing private endpoints and subscription feeds that need authentication make sure that the `Access WebSockets API` API key permission is set in the users Kraken account.

Please note that this function automatically assigns the `method` key and sets its value to `subscribe`. The authentication token is also assigned automatically, so only the `params` are needed here.

Parameters

- **params** (*dict*) – The subscription message
- **req_id** (*int, optional*) – Identification number that will be added to the response message sent by the websocket feed.

Initialize your client as described in *kraken.spot.SpotWSClient* to run the following example:

Listing 25: Spot Websocket: Subscribe to a websocket feed

```
>>> await client.subscribe(
...     params={"channel": "ticker", "symbol": ["BTC/USD"]}
... )
```

async unsubscribe(*params: dict*, *req_id: int | None = None*) → None

Unsubscribe from a channel/feed

Success or failures are sent via the websocket connection and can be received via the on_message or callback function.

When accessing private endpoints and subscription feeds that need authentication make sure, that the `Access WebSockets API` API key permission is set in the users Kraken account.

Parameters

params (*dict*) – The un-subscription message (only the params part)

Initialize your client as described in *kraken.spot.SpotWSClient* to run the following example:

Listing 26: Spot Websocket: Unsubscribe from a websocket feed

```
>>> await client.unsubscribe(
...     params={"channel": "ticker", "symbol": ["BTC/USD"]}
... )
```

CHAPTER

EIGHT

FUTURES REST CLIENTS

class kraken.base_api.**FuturesClient**(*key: str = ''*, *secret: str = ''*, *url: str = ''*, *proxy: str | None = None*, *, *sandbox: bool = False*, *use_custom_exceptions: bool = True*)

Bases: `object`

The base class for all Futures clients handles un-/signed requests and returns exception handled results.

If you are facing timeout errors on derived clients, you can make use of the `TIMEOUT` attribute to deviate from the default `10` seconds.

If the sandbox environment is chosen, the keys must be generated from here:
https://demo-futures.kraken.com/settings/api

Kraken sometimes rejects requests that are older than a certain time without further information. To avoid this, the session manager creates a new session every 5 minutes.

Parameters

- **key** (*str, optional*) – Futures API public key (default: "")
- **secret** (*str, optional*) – Futures API secret key (default: "")
- **url** (*str, optional*) – The URL to access the Futures Kraken API (default: https://futures.kraken.com)
- **sandbox** (*bool, optional*) – If set to `True` the URL will be https://demo-futures.kraken.com (default: `False`)
- **proxy** (*str, optional*) – proxy URL, may contain authentication information

get_nonce() → str

Return a new nonce

request(*method: str*, *uri: str*, *post_params: dict | None = None*, *query_params: dict | None = None*, *timeout: int = 10*, *, *auth: bool = True*, *return_raw: bool = False*, *extra_params: str | dict | None = None*) → dict | list | Response

Handles the requested requests, by sending the request, handling the response, and returning the message or in case of an error the respective Exception.

Parameters

- **method** (*str*) – The request method, e.g., `GET`, `POST`, and `PUT`
- **uri** (*str*) – The endpoint to send the message
- **post_params** (*dict, optional*) – The query parameter of the request (default: `None`)
- **extra_params** (*str | dict, optional*) – Additional query parameter of the request (default: `None`)

- **query_params** (*dict, optional*) – The query parameter of the request (default: None)
- **timeout** (*int*) – Timeout for the request (default: 10)
- **auth** (*bool*) – If the request needs authentication (default: True)
- **return_raw** (*bool, optional*) – If the response should be returned without parsing. This is used for example when requesting an export of the trade history as .zip archive.

Raises
kraken.exceptions.* – If the response contains errors

Returns
The response

Return type
dict | list | requests.Response

class kraken.base_api.**FuturesAsyncClient**(*key: str = ''*, *secret: str = ''*, *url: str = ''*, *proxy: str | None = None*, ***, *sandbox: bool = False*, *use_custom_exceptions: bool = True*)

Bases: *FuturesClient*

This class provides the base client for accessing the Kraken Futures API using asynchronous requests.

If you are facing timeout errors on derived clients, you can make use of the TIMEOUT attribute to deviate from the default 10 seconds.

If the sandbox environment is chosen, the keys must be generated from here:
https://demo-futures.kraken.com/settings/api

Parameters

- **key** (*str, optional*) – Futures API public key (default: "")
- **secret** (*str, optional*) – Futures API secret key (default: "")
- **url** (*str, optional*) – The URL to access the Futures Kraken API (default: https://futures.kraken.com)
- **proxy** (*str, optional*) – proxy URL, may contain authentication information
- **sandbox** (*bool, optional*) – If set to True the URL will be https://demo-futures.kraken.com (default: False)

async_close() → None

Closes the aiohttp session

async close() → None

Closes the aiohttp session

get_nonce() → str

Return a new nonce

async request(*method: str*, *uri: str*, *post_params: dict | None = None*, *query_params: dict | None = None*, *timeout: int = 10*, ***, *auth: bool = True*, *return_raw: bool = False*) → dict | list | aiohttp.ClientResponse | Awaitable

Handles the requested requests, by sending the request, handling the response, and returning the message or in case of an error the respective Exception.

Parameters

- **method** (*str*) – The request method, e.g., GET, POST, and PUT

- **uri** (*str*) – The endpoint to send the message
- **post_params** (*dict, optional*) – The query parameter of the request (default: None)
- **extra_params** (*str | dict, optional*) – Additional query parameter of the request (default: None)
- **query_params** (*dict, optional*) – The query parameter of the request (default: None)
- **timeout** (*int*) – Timeout for the request (default: 10)
- **auth** (*bool*) – If the request needs authentication (default: True)
- **return_raw** (*bool, optional*) – If the response should be returned without parsing. This is used for example when requesting an export of the trade history as .zip archive.

Raises
kraken.exceptions.* – If the response contains errors

Returns
The response

Return type
dict | list | requests.Response

class kraken.futures.**User**(*key: str = ''*, *secret: str = ''*, *url: str = ''*, *proxy: str | None = None*, ***, *sandbox: bool = False*)

Bases: *FuturesClient*

Class that implements the Kraken Futures user client

If the sandbox environment is chosen, the keys must be generated from here: https://demo-futures.kraken.com/settings/api

Parameters

- **key** (*str, optional*) – Futures API public key (default: "")
- **secret** (*str, optional*) – Futures API secret key (default: "")
- **url** (*str, optional*) – Alternative URL to access the Futures Kraken API (default: https://futures.kraken.com)
- **proxy** (*str, optional*) – proxy URL, may contain authentication information
- **sandbox** (*bool, optional*) – If set to True the URL will be https://demo-futures.kraken.com (default: False)

Listing 1: Futures User: Create the user client

```
>>> from kraken.futures import User
>>> user = User() # unauthenticated
>>> user = User(key="api-key", secret="secret-key") # authenticated
```

Listing 2: Futures User: Create the user client as context manager

```
>>> from kraken.futures import User
>>> with User(key="api-key", secret="secret-key") as user:
...     print(user.get_wallets())
```

check_trading_enabled_on_subaccount(*subaccountUid: str*, ***, *extra_params: dict | None = None*) → dict

Checks if trading is enabled or disabled on the specified subaccount.

Requires the `General API - Full Access` permission in the API key settings.

This endpoint is only available for institutional clients and is not tested so far and results in `KrakenAuthenticationError`.

- https://docs.kraken.com/api/docs/futures-api/trading/get-subaccount-trading-capability

Returns
The open futures positions/orders

Return type
dict

Listing 3: Futures User: Check if trading is enabled on a subaccount

```
>>> from kraken.futures import User
>>> user = User(key="api-key", secret="secret-key")
>>> user.set_trading_on_subaccount(
...     subaccountUid="778387bh61b-f990-4128-16a7-f4ab669a9b",
... )
{
   "tradingEnabled": False
}
```

get_account_log(*before: str | int | None = None*, *count: str | int | None = None*, *from_: str | int | None = None*, *info: str | None = None*, *since: str | int | None = None*, *sort: str | None = None*, *to: str | None = None*, ***, *extra_params: dict | None = None*) → dict

Get the historical events of the user's account. This is not available in the Kraken demo/sandbox environment.

Requires at least the `General API - Read Only` permission in the API key settings.

- https://docs.kraken.com/api/docs/futures-api/history/account-log

Parameters

- **before** (*str | int, optional*) – Filter to only return results before a specific timestamp or date
- **count** (*str | int, optional*) – Defines the maximum number of results (max: `500`)
- **from** (*str | int, optional*) – Defines the first id to start with
- **info** (*str, optional*) – Filter by info (e.g.,: `futures liquidation`)
- **since** (*str | int, optional*) – Defines the first entry to begin with by item
- **sort** (*str, optional*) – Sort the results
- **to** (*str, optional*) – Id of the last entry

Returns
The account log

Return type
dict

Listing 4: Futures User: Get the user's account log

```
>>> from kraken.futures import User
>>> user = User(key="api-key", secret="secret-key")
>>> user.get_account_log(before="2023-04-04T16:10:46.260Z", count=1)
{
    'accountUid': 'f7d5571c-6d10-4cf1-944a-048d25682ed0',
    'logs': [
        {
            'asset': 'eth',
            'contract': None,
            'booking_uid': 'be3d0e19-887a-4a8e-b8f6-32b9ef7f04ab',
            'collateral': None,
            'date': '2023-04-04T16:10:46.260Z',
            'execution': None,
            'fee': None,
            'funding_rate': None,
            'id': 10,
            'info': 'admin transfer',
            'margin_account': 'ETH',
            'mark_price': None,
            'new_average_entry_price': None,
            'new_balance': 2.6868286287,
            'old_average_entry_price': None,
            'old_balance': 0.0,
            'realized_funding': None,
            'realized_pnl': None,
            'trade_price': None,
            'conversion_spread_percentage': None
        }
    ], ...
}
```

get_account_log_csv(*, *extra_params: dict | None = None*) → requests.Response

Return the account log as csv, for example to export it. This is not available in the Kraken demo/sandbox environment.

Requires at least the `General API - Read Only` permission in the API key settings.

- https://docs.kraken.com/api/docs/futures-api/history/account-log-csv

Listing 5: Futures User: Get the account log and export as CSV

```
>>> from kraken.futures import User
>>> user = User(key="api-key", secret="secret-key")
>>> response = user.get_account_log_csv()
>>> with open(f"account_log.csv", "wb") as file:
...     for chunk in response.iter_content(chunk_size=512):
...         if chunk:
...             file.write(chunk)
```

get_execution_events(*before: int | None = None*, *continuation_token: str | None = None*, *since: int | None = None*, *sort: str | None = None*, *tradeable: str | None = None*, *, *extra_params: dict | None = None*) → dict

Retrieve the order/position execution events of this user. The returned `continuation_token` can be used to request more data.

Requires at least the `General API - Read Only` permission in the API key settings.

- https://docs.kraken.com/api/docs/futures-api/history/get-execution-events

(If you are facing some timeout error, just set the clients attribute `TIMEOUT` temporarily to the desired amount in seconds.)

Parameters

- **before** (*int, optional*) – Filter by time
- **continuation_token** (*str, optional*) – Token that can be used to continue requesting historical events
- **since** (*int, optional*) – Filter by a specifying a start point
- **sort** (*str, optional*) – Sort the results
- **tradeable** (*str, optional*) – The asset to filter for

Returns

The user-specific execution events

Return type

dict

Listing 6: Futures User: Get the user's historical execution events

```
>>> from kraken.futures import User
>>> user = User(key="api-key", secret="secret-key")
>>> user.get_execution_events(
...     tradeable="PF_SOLUSD",
...     since=1668989233,
...     before=1668999999,
...     sort="asc"
... )
{
    'accountUid': 'f7d5571c-6d10-4cf1-944a-048d25682ed0',
    'continuationToken': 'alp81a',
    'elements': [{
        'event': {
            'execution': {
                'execution': {
                    'limitFilled': false,
                    'makerOrder': {
                        'accountUid': 'f7d5571c-6d10-4cf1-944a-048d25682ed0',
                        'direction': 'Buy',
                        'filled': '2332.12239',
                        'lastUpdateTimestamp': 1605126171852,
                        'limitPrice': '1234.56789',
                        'orderType': 'lmt',
                        'quantity': '1234.56789',
                        'reduceOnly': false,
                        'timestamp': 1605126171852,
                        'tradeable': 'pi_xbtusd',
                        'uid': 'f7d5571c-6d10-4cf1-944a-048d25682ed0'
```

(continues on next page)

(continued from previous page)

```
                    },
                    'makerOrderData': {
                        'fee': '12.56789',
                        'positionSize': '2332.12239'
                    },
                    'markPrice': '27001.56',
                    'oldTakerOrder': {
                        'accountUid': 'f7d5571c-6d10-4cf1-944a-048d25682ed0',
                        'direction': 'Buy',
                        'filled': '2332.12239',
                        'lastUpdateTimestamp': 1605126171852,
                        'limitPrice': '27002.56789',
                        'orderType': 'string',
                        'quantity': '0.156789',
                        'reduceOnly': false,
                        'timestamp': 1605126171852,
                        'tradeable': 'pi_xbtusd',
                        'uid': 'f7d5571c-6d10-4cf1-944a-048d25682ed0'
                    },
                    'price': '2701.8163',
                    'quantity': '0.156121',
                    'takerOrder': {
                        'accountUid': 'f7d5571c-6d10-4cf1-944a-048d25682ed0',
                        'direction': 'Buy',
                        'filled': '0.156121',
                        'lastUpdateTimestamp': 1605126171852,
                        'limitPrice': '2702.91',
                        'orderType': 'lmt',
                        'quantity': '0.156121',
                        'reduceOnly': false,
                        'timestamp': 1605126171852,
                        'tradeable': 'pi_xbtusd',
                        'uid': 'f7d5571c-6d10-4cf1-944a-048d25682ed0'
                    },
                    'takerOrderData': {
                        'fee': '12.83671',
                        'positionSize': '27012.91'
                    },
                    'timestamp': 1605126171852,
                    'uid': 'f7d5571c-6d10-4cf1-944a-048d25682ed0',
                    'usdValue': '2301.56789'
                },
            }
        },
        'timestamp': 1605126171852,
        'uid': 'f7d5571c-6d10-4cf1-944a-048d25682ed0'
    }, ...],
    'len': 0,
    'serverTime': '2023-04-06T21:11:31.677Z'
}
```

get_nonce() → str

Return a new nonce

get_notifications(*, *extra_params: dict | None = None*) → dict

Retrieve the latest notifications from the Kraken Futures API

Requires at least the `General API - Read Only` permission in the API key settings.

- https://docs.kraken.com/api/docs/futures-api/trading/get-notifications

Returns
Notifications

Return type
dict

Listing 7: Futures User: Get the latest notifications

```
>>> from kraken.futures import User
>>> user = User(key="api-key", secret="secret-key")
>>> user.get_notifications()
{
   'result': 'success',
    'notifications': [{
        'type': 'general',
        'priority': 'high',
        'note': 'Market in post only mode until 4pm.'
    }],
    'serverTime': '2023-04-04T18:01:39.729Z'
}
```

get_open_orders(*, *extra_params: dict | None = None*) → dict

Retrieve the open orders.

Requires at least the `General API - Read Only` permission in the API key settings.

- https://docs.kraken.com/api/docs/futures-api/trading/get-open-orders

Returns
The open futures positions/orders

Return type
dict

Listing 8: Futures User: Get open orders

```
>>> from kraken.futures import User
>>> user = User(key="api-key", secret="secret-key")
>>> user.get_open_orders()
{
    'result': 'success',
    'openOrders': [{
        'order_id': '2ce038ae-c144-4de7-a0f1-82f7f4fca864',
        'symbol': 'pi_ethusd',
        'side': 'buy',
        'orderType': 'lmt',
        'limitPrice': 1200,
```

(continues on next page)

(continued from previous page)

```
        'unfilledSize': 100,
        'receivedTime': '2023-04-07T15:18:04.699Z',
        'status': 'untouched',
        'filledSize': 0,
        'reduceOnly': False,
        'lastUpdateTime': '2023-04-07T15:18:04.699Z'
    }, {
        'order_id': 'c8135f52-2a86-4e26-b629-43cc37da9dbf',
        'symbol': 'pi_ethusd',
        'side': 'buy',
        'orderType': 'take_profit',
        'limitPrice': 1860,
        'stopPrice': 1880.4,
        'unfilledSize': 10,
        'receivedTime': '2023-04-07T15:14:25.995Z',
        'status': 'untouched',
        'filledSize': 0,
        'reduceOnly': False,
        'triggerSignal': 'last',
        'lastUpdateTime': '2023-04-07T15:14:25.995Z'
    }, {
        'order_id': 'e58ed100-1fb8-4e6c-a5ea-1cf85b0f0654',
        'symbol': 'pi_ethusd',
        'side': 'buy',
        'orderType': 'take_profit',
        'limitPrice': 1860,
        'stopPrice': 1880.4,
        'unfilledSize': 10,
        'receivedTime': '2023-04-07T15:12:08.131Z',
        'status': 'untouched',
        'filledSize': 0,
        'reduceOnly': False,
        'triggerSignal': 'last',
        'lastUpdateTime': '2023-04-07T15:12:08.131Z'
    }, {
        'order_id': 'c8776f6e-c29e-4c6a-83ee-2d3cc6781cda',
        'symbol': 'pf_ethusd',
        'side': 'buy',
        'orderType': 'take_profit',
        'limitPrice': 1860,
        'stopPrice': 5,
        'unfilledSize': 0.5,
        'receivedTime': '2023-04-07T14:57:37.849Z',
        'status': 'untouched',
        'filledSize': 0,
        'reduceOnly': True,
        'triggerSignal': 'last',
        'lastUpdateTime': '2023-04-07T14:57:37.849Z'
    }, ...],
    'serverTime': '2023-04-07T15:30:29.911Z'
}
```

get_open_positions(*, *extra_params: dict | None = None*) → dict

List the open positions of the user.

Requires at least the `General API - Read Only` permission in the API key settings.

- https://docs.kraken.com/api/docs/futures-api/trading/get-open-positions

Returns
Information about the open positions

Return type
dict

Listing 9: Futures User: Get the user's open positions

```
>>> from kraken.futures import User
>>> user = User(key="api-key", secret="secret-key")
>>> user.get_open_positions()
{
    'result': 'success',
    'openPositions': [
        {
            'side': 'short',
            'symbol': 'pi_xbtusd',
            'price': 27523.749993345933,
            'fillTime': '2023-04-05T12:31:21.410Z',
            'size': 8000,
            'unrealizedFunding': 0.00005879463852989987
        },
    ],
    'serverTime': '2023-04-06T16:12:15.410Z'
}
```

get_order_events(*before: int | None = None*, *continuation_token: str | None = None*, *since: int | None = None*, *sort: str | None = None*, *tradeable: str | None = None*, ***, *extra_params: dict | None = None*) → dict

Retrieve information about the user-specific order events including opened, closed, filled, etc. The returned `continuation_token` can be used to request more data.

Requires at least the `General API - Read Only` permission in the API key settings.

- https://docs.kraken.com/api/docs/futures-api/history/get-order-events

Parameters

- **before** (*int, optional*) – Filter by time
- **continuation_token** (*str, optional*) – Token that can be used to continue requesting historical events
- **since** (*int, optional*) – Filter by a specifying a start point
- **sort** (*str, optional*) – Sort the results
- **tradeable** (*str, optional*) – The asset to filter for

Returns
The user-specific order events

Return type
dict

Listing 10: Futures User: Get the user's historical order events

```
>>> from kraken.futures import User
>>> user = User(key="api-key", secret="secret-key")
>>> user.get_order_events(tradeable="PF_SOLUSD", since=1668989233,␣
↪before=1668999999, sort="asc")
{
    'accountUid': 'f7d5571c-6d10-4cf1-944a-048d25682ed0',
    'continuationToken': 'simb178',
    'elements': [{
        'event': {
            'OrderPlaced': {
                'order': {
                    'accountUid': 'f7d5571c-6d10-4cf1-944a-048d25682ed0',
                    'direction': 'Sell',
                    'filled': '12.011',
                    'lastUpdateTimestamp': 1605126171852,
                    'limitPrice': '28900.0',
                    'orderType': 'string',
                    'quantity': '13.12',
                    'reduceOnly': false,
                    'timestamp': 1605126171852,
                    'tradeable': 'pi_xbtusd',
                    'uid': 'f7d5571c-6d10-4cf1-944a-048d25682ed0'
                },
                'reason': 'string',
                'reducedQuantity': 'string'
            }
        },
        'timestamp': 1605126171852,
        'uid': 'f7d5571c-6d10-4cf1-944a-048d25682ed0'
    }, ...],
    'len': 10,
    'serverTime': '2023-04-05T12:31:42.677Z'
}
```

get_subaccounts(*, *extra_params: dict | None = None*) → dict

List the subaccounts of the user.

Requires at least the `General API - Read Only` permission in the API key settings.

- https://docs.kraken.com/api/docs/futures-api/trading/list-subaccounts

Returns
Information about the user owned subaccounts

Return type
dict

Listing 11: Futures User: Get the user's subaccounts

```
>>> from kraken.futures import User
>>> user = User(key="api-key", secret="secret-key")
>>> user.get_subaccounts()
{
    'result': 'success',
    'serverTime': '2023-04-04T18:03:33.696Z',
    'masterAccountUid': 'f7d5571c-6d10-4cf1-944a-048d25682ed0',
    'subaccounts': []
}
```

get_trigger_events(*before: int | None = None*, *continuation_token: str | None = None*, *since: int | None = None*, *sort: str | None = None*, *tradeable: str | None = None*, ***, *extra_params: dict | None = None*) → dict

Retrieve information about trigger events.

The returned `continuation_token` can be used to request more data.

Requires at least the `General API - Read Only` permission in the API key settings.

- https://docs.kraken.com/api/docs/futures-api/history/get-trigger-events

Parameters

- **before** (*int, optional*) – Filter by time
- **continuation_token** (*str, optional*) – Token that can be used to continue requesting historical events
- **since** (*int, optional*) – Filter by a specifying a start point
- **sort** (*str, optional*) – Sort the results
- **tradeable** (*str, optional*) – The asset to filter for

Returns
The user-specific trigger events

Return type
dict

Listing 12: Futures User: Get the user's historical trigger events

```
>>> from kraken.futures import User
>>> user = User(key="api-key", secret="secret-key")
>>> user.get_trigger_events(
...     tradeable="PF_SOLUSD",
...     since=1668989233,
...     before=1668999999,
...     sort="asc"
... )
{
    "accountUid": "f7d5571c-6d10-4cf1-944a-048d25682ed0",
    "continuationToken": "c3RyaW5n",
    "elements": [{
        "event": {
```

(continues on next page)

(continued from previous page)

```
            "OrderTriggerPlaced": {
                "order": {
                    "accountId": 0.0,
                    "accountUid": "f7d5571c-6d10-4cf1-944a-048d25682ed0",
                    "direction": "Buy",
                    "lastUpdateTimestamp": 1605126171852,
                    "limitPrice": "29000.0",
                    "orderType": "lmt",
                    "quantity": "1.0",
                    "reduceOnly": false,
                    "timestamp": 1605126171852,
                    "tradeable": "pi_xbtusd",
                    "triggerOptions": {
                        "trailingStopOptions": {
                            "maxDeviation": "0.1",
                            "unit": "Percent"
                        },
                        "triggerPrice": "29200.0",
                        "triggerSide": "Sell",
                        "triggerSignal": "trade"
                    },
                    "uid": "f7d5571c-6d10-4cf1-944a-048d25682ed0"
                },
                "reason": "maxDeviation triggered"
            }
        },
        "timestamp": 1605126171852,
        "uid": "f7d5571c-6d10-4cf1-944a-048d25682ed0"
    }, ...],
    "len": 10,
    "serverTime": "2022-03-31T20:38:53.677Z"
}
```

get_unwind_queue(*, *extra_params: dict | None = None*) → dict

Retrieve information about the percentile of the open position in case of unwinding.

Requires at least the `General API - Read Only` permission in the API key settings.

- https://docs.kraken.com/api/docs/futures-api/trading/get-unwind-queue

Returns
Information about unwind queue

Return type
dict

Listing 13: Futures User: Get the user's unwind queue

```
>>> from kraken.futures import User
>>> user = User(key="api-key", secret="secret-key")
>>> user.get_unwind_queue()
{
    'result': 'success',
```

(continues on next page)

(continued from previous page)

```
    'serverTime': '2023-04-04T18:05:01.328Z',
    'queue': [
        { 'symbol': 'PF_UNIUSD', 'percentile': 20 }
    ]
}
```

get_wallets(*, *extra_params: dict | None = None*) → dict

Lists the current wallet balances of the user.

Requires at least the `General API - Read Only` permission in the API key settings.

- https://docs.kraken.com/api/docs/futures-api/trading/get-accounts

Returns
Information about the current balances of the user

Return type
dict

Listing 14: Futures User: Get the user's wallets

```
>>> from kraken.futures import User
>>> user = User(key="api-key", secret="secret-key")
>>> user.get_wallets()
{
    'result': 'success',
    'accounts': {
        'fi_xbtusd': {
            'auxiliary': {
                'usd': 0,
                'pv': 0.0,
                'pnl': 0.0,
                'af': 0.0,
                'funding': 0.0
            },
            'marginRequirements': {
                'im': 0.0,
                'mm': 0.0,
                'lt': 0.0,
                'tt': 0.0
            },
            'triggerEstimates': {
                'im': 0,
                'mm': 0,
                'lt': 0,
                'tt': 0
            },
            'balances': {
                'xbt': 0.0
            },
            'currency': 'xbt',
            'type': 'marginAccount'
        },
```

(continues on next page)

(continued from previous page)

```
        'cash': {
            'balances': {
                'eur': 4567.7117591172,
                'gbp': 4002.4975584765,
                'bch': 39.3081761006,
                'usd': 5000.0,
                'xrp': 10055.1019587339,
                'eth': 2.6868286287,
                'usdt': 4999.3200924674,
                'usdc': 4999.8300057798,
                'ltc': 53.9199827456,
                'xbt': 0.1785169809
            },
            'type': 'cashAccount'
        },
        'flex': {
            'currencies': {},
            'initialMargin': 0.0,
            'initialMarginWithOrders': 0.0,
            'maintenanceMargin': 0.0,
            'balanceValue': 0.0,
            'portfolioValue': 0.0,
            'collateralValue': 0.0,
            'pnl': 0.0,
            'unrealizedFunding': 0.0,
            'totalUnrealized': 0.0,
            'totalUnrealizedAsMargin': 0.0,
            'availableMargin': 0.0,
            'marginEquity': 0.0,
            'type': 'multiCollateralMarginAccount'
        }
    },
    'serverTime': '2023-04-04T17:56:49.027Z'
}
```

request(*method: str*, *uri: str*, *post_params: dict | None = None*, *query_params: dict | None = None*, *timeout: int = 10*, ***, *auth: bool = True*, *return_raw: bool = False*, *extra_params: str | dict | None = None*) → dict | list | Response

Handles the requested requests, by sending the request, handling the response, and returning the message or in case of an error the respective Exception.

Parameters

- **method** (*str*) – The request method, e.g., `GET`, `POST`, and `PUT`
- **uri** (*str*) – The endpoint to send the message
- **post_params** (*dict, optional*) – The query parameter of the request (default: `None`)
- **extra_params** (*str | dict, optional*) – Additional query parameter of the request (default: `None`)
- **query_params** (*dict, optional*) – The query parameter of the request (default: `None`)
- **timeout** (*int*) – Timeout for the request (default: `10`)

- **auth** (*bool*) – If the request needs authentication (default: `True`)
- **return_raw** (*bool, optional*) – If the response should be returned without parsing. This is used for example when requesting an export of the trade history as .zip archive.

Raises
kraken.exceptions.* – If the response contains errors

Returns
The response

Return type
dict | list | requests.Response

set_trading_on_subaccount(*subaccountUid: str*, ***, *trading_enabled: bool*, *extra_params: dict | None = None*) → dict

Enable or disable trading on a subaccount.

Requires the `General API - Full Access` permission in the API key settings.

This endpoint is only available for institutional clients and is not tested so far and always results in `INTERNAL_SERVER_ERROR`.

- https://docs.kraken.com/api/docs/futures-api/trading/update-subaccount-trading-capability

Returns
The open futures positions/orders

Return type
dict

Listing 15: Futures User: Dis-/Enable trading on a subaccount

```
>>> from kraken.futures import User
>>> user = User(key="api-key", secret="secret-key")
>>> user.set_trading_on_subaccount(
...     subaccountUid="778387bh61b-f990-4128-16a7-f4ab669a9b",
...     trading_enabled=True
... )
{
    "tradingEnabled": True
}
```

class kraken.futures.**Market**(*key: str = ''*, *secret: str = ''*, *url: str = ''*, *proxy: str | None = None*, ***, *sandbox: bool = False*)

Bases: *FuturesClient*

Class that implements the Kraken Futures market client

If the sandbox environment is chosen, the keys must be generated from here: https://demo-futures.kraken.com/settings/api

Parameters

- **key** (*str, optional*) – Futures API public key (default: "")
- **secret** (*str, optional*) – Futures API secret key (default: "")
- **url** (*str, optional*) – Alternative URL to access the Futures Kraken API (default: https://futures.kraken.com)

- **proxy** (*str, optional*) – proxy URL, may contain authentication information
- **sandbox** (*bool, optional*) – If set to `True` the URL will be https://demo-futures.kraken.com (default: `False`)

Listing 16: Futures Market: Create the market client

```
>>> from kraken.futures import Market
>>> market = Market() # unauthenticated
>>> market = Market(key="api-key", secret="secret-key") # authenticated
```

Listing 17: Futures Market: Create the market client as context manager

```
>>> from kraken.futures import Market
>>> with Market() as market:
...     print(market.get_tick_types())
```

get_fee_schedules(*, *extra_params: dict | None = None*) → dict

Retrieve information about the current fees

- https://docs.kraken.com/api/docs/futures-api/trading/get-fee-schedules-v-3
- https://www.kraken.com/features/fee-schedule

This function uses caching. Run `get_fee_schedules.cache_clear()` to clear.

Returns
Dictionary containing information about the fees for wide range of tradeable assets

Return type
dict

Listing 18: Futures Market: Get the available fee schedules

```
>>> from kraken.futures import Market
>>> Market().get_fee_schedules()
{
    'feeSchedules': [{
        'uid': '5b755fea-c5b0-4307-a66e-b392cd5bd931',
        'name': 'KF USD Multi-Collateral Fees',
        'tiers': [
            {'makerFee': 0.02, 'takerFee': 0.05, 'usdVolume': 0.0},
            {'makerFee': 0.015, 'takerFee': 0.04, 'usdVolume': 100000.0},
            {'makerFee': 0.0125, 'takerFee': 0.03, 'usdVolume': 1000000.0},
            {'makerFee': 0.01, 'takerFee': 0.025, 'usdVolume': 5000000.0},
            {'makerFee': 0.0075, 'takerFee': 0.02, 'usdVolume': 10000000.0},
            {'makerFee': 0.005, 'takerFee': 0.015, 'usdVolume': 20000000.0},
            {'makerFee': 0.0025, 'takerFee': 0.0125, 'usdVolume': 50000000.0},
            {'makerFee': 0.0, 'takerFee': 0.01, 'usdVolume': 100000000.0}
        ]}, ...
    ]
}
```

get_fee_schedules_vol(*, *extra_params: dict | None = None*) → dict

Get the personal volumes per fee schedule

Requires the `General API - Full Access` permission in the API key settings.

- https://docs.kraken.com/api/docs/futures-api/trading/get-user-fee-schedule-volumes-v-3

Listing 19: Futures Market: Get the personal fee schedule volumes

```
>>> from kraken.futures import Market
>>> market = Market(key="api-key", secret="secret-key")
>>> market.get_fee_schedules_vol()
{
    'volumesByFeeSchedule': {
        'ffb5403d-e82e-4ef0-8792-86a0471f526a': 0,
        '5b755fea-c5b0-4307-a66e-b392cd5bd931': 0,
        'eef90775-995b-4596-9257-0917f6134766': 0
    }
}
```

get_historical_funding_rates(*symbol: str*, *, *extra_params: dict | None = None*) → dict

Retrieve information about the historical funding rates for a specific asset.

- https://docs.kraken.com/api/docs/futures-api/trading/historical-funding-rates

Parameters
symbol (*str*) – The symbol/asset/futures contract

Returns
Funding rates

Return type
dict

Listing 20: Futures Market: Get the historical funding rates

```
>>> from kraken.futures import Market
>>> Market().get_historical_funding_rates(symbol="PI_XBTUSD")
{
    'rates': [{
            'timestamp': '2018-08-31T16:00:00.000Z',
            'fundingRate': 1.0327058177e-08,
            'relativeFundingRate': 7.182407e-05
        }, {
            'timestamp': '2018-08-31T20:00:00.000Z',
            'fundingRate': -1.2047162502e-08,
            'relativeFundingRate': -8.4873103125e-05
        }, {
            'timestamp': '2018-09-01T00:00:00.000Z',
            'fundingRate': -9.645113378e-09,
            'relativeFundingRate': -6.76651e-05
        }, ...
    ]
}
```

get_instruments(*, *extra_params: dict | None = None*) → dict

Retrieve more specific information about the tradeable assets on the Futures market

- https://docs.kraken.com/api/docs/futures-api/trading/get-instruments

Returns
Dictionary containing information for all tradeable assets

Return type
dict

Listing 21: Futures Market: Get the available instruments/assets and information

```
>>> from kraken.futures import Market
>>> Market().get_instruments()
{
  'instruments': [{
        'symbol': 'pi_xbtusd',
        'type': 'futures_inverse',
        'underlying': 'rr_xbtusd',
        'tickSize': 0.5,
        'contractSize': 1,
        'tradeable': True,
        'impactMidSize': 1000.0,
        'maxPositionSize': 75000000.0,
        'openingDate': '2018-08-31T00:00:00.000Z',
        'marginLevels': [{
                'contracts': 0,
                'initialMargin': 0.02,
                'maintenanceMargin': 0.01
            }, {
                'contracts': 500000,
                'initialMargin': 0.04,
                'maintenanceMargin': 0.02
            }, {
                'contracts': 1000000,
                'initialMargin': 0.06,
                'maintenanceMargin': 0.03
            }, {
                'contracts': 3000000,
                'initialMargin': 0.1,
                'maintenanceMargin': 0.05
            }, {
                'contracts': 6000000,
                'initialMargin': 0.15,
                'maintenanceMargin': 0.075
            }, {
                'contracts': 12000000,
                'initialMargin': 0.25,
                'maintenanceMargin': 0.125
            }, {
                'contracts': 20000000,
                'initialMargin': 0.3,
                'maintenanceMargin': 0.15
            }, {
                'contracts': 50000000,
                'initialMargin': 0.4,
```

(continues on next page)

(continued from previous page)

```
                'maintenanceMargin': 0.2
            }
        ],
        'fundingRateCoefficient': 24,
        'maxRelativeFundingRate': 0.0025,
        'isin': 'GB00J62YGL67',
        'contractValueTradePrecision': 0,
        'postOnly': False,
        'feeScheduleUid': 'eef90775-995b-4596-9257-0917f6134766',
        'retailMarginLevels': [{
            'contracts': 0,
            'initialMargin': 0.5,
            'maintenanceMargin': 0.25
        }],
        'category': '', 'tags': []
    }
}
```

get_instruments_status(*instrument: str | None = None*, *, *extra_params: dict | None = None*) → dict

Retrieve status information of a specific or all futures contracts.

- https://docs.kraken.com/api/docs/futures-api/trading/instrument-status

Parameters
instrument (*str | None, optional*) – Filter by asset

Returns
Status information about the asset(s)

Return type
dict

Listing 22: Futures Market: Retrieve information about a specific asset/contract/instrument

```
>>> from kraken.futures import Market
>>> Market().get_instruments_status(instrument="PI_XBTUSD")
{
    'tradeable': 'PI_XBTUSD',
    'experiencingDislocation': False,
    'priceDislocationDirection': None,
    'experiencingExtremeVolatility': False,
    'extremeVolatilityInitialMarginMultiplier': 1
}
```

get_leverage_preference(*, *extra_params: dict | None = None*) → dict

Get the current leverage preferences of the user.

Requires at least the `General API - Read Only` permission in the API key settings.

- https://docs.kraken.com/api/docs/futures-api/trading/get-leverage-setting

Returns
The leverage preferences for all futures contracts

Return type
dict

Listing 23: Futures Market: Get the users leverage preferences

```
>>> from kraken.futures import Market
>>> market = Market(key="api-key", secret="secret-key")
>>> market.get_leverage_preference()
{
    'leveragePreferences': [
        {'symbol': 'PF_XBTUSD', 'maxLeverage': 5.0},
        ...
    ]
}
```

get_nonce() → str

Return a new nonce

get_ohlc(*tick_type: str*, *symbol: str*, *resolution: int*, *from_: int | None = None*, *to: int | None = None*, ***, *extra_params: dict | None = None*) → dict

Retrieve the open, high, low, and close data for a specific symbol and resolution. It is also possible to filter by time.

- https://docs.kraken.com/api/docs/futures-api/charts/candles

Parameters

- **tick_type** (*str*) – The kind of data, based on `mark`, `spot`, or `trade`
- **symbol** (*str*) – The asset pair to get the ohlc from
- **resolution** (*str*) – The tick resolution, one of `1m`. `5m`, `15m`, `1h`, `4h`, `12h`, `1d`, `1w`
- **from** (*int, optional*) – From date in epoch seconds
- **to** (*int, optional*) – To date in epoch seconds (inclusive)

Returns
The current OHLC data for a specific asset pair

Return type
dict

Listing 24: Futures Market: Get the OHLC data

```
>>> from kraken.futures import Market
>>> Market().get_ohlc(tick_type="trade", symbol="PI_XBTUSD", resolution="1h")
{
    'candles': [
        {
            'time': 1680624000000,
            'open': '28050.0',
            'high': '28150',
            'low': '27983.0',
            'close': '28126.0',
            'volume': '1089794.00000000'
        }
```

(continues on next page)

(continued from previous page)

```
    ],
    'more_candles': True
}
```

get_orderbook(*symbol: str | None = None*, *, *extra_params: dict | None = None*) → dict

Get the orderbook of a specific asset/symbol. Even if the official kraken documentation states that the parameter `symbol` is not required, they will always respond with an error message, so it is recommended to use the `symbol` parameter until they don't fix this issue.

- https://docs.kraken.com/api/docs/futures-api/trading/get-orderbook

Parameters
symbol (*str, optional*) – The asset/symbol to get the orderbook from

Returns
The current orderbook for the futures contracts

Return type
dict

Listing 25: Futures Market: Get the assets orderbook

```
>>> from kraken.futures import Market
>>> Market().get_orderbook(symbol="PI_XBTUSD")
{
    'result': 'success', 'orderBook': {
        'bids': [
            [27909, 3000], [27908.5, 1703], [27906, 1716],
            [27905, 2900], [27904.5, 2900], [27904, 2900],
            [27903.5, 8900], [27903, 3415], [27902.5, 2900],
            [27902, 2900], [27901, 4200], [27900.5, 6000],
            ...
        ],
        'asks': [
            [27915, 4200], [27916, 1706], [27917.5, 2900],
            [27918, 4619], [27918.5, 4200], [27919, 2900],
            [27919.5, 78], [27920, 2900], [27920.5, 2900],
            [27921, 6342], [27921.5, 4200], [27923, 27851],
            ...
        ]
    }
}
```

get_pnl_preference(*, *extra_params: dict | None = None*) → dict

Get the current PNL (profit & loss) preferences. This can be used to define the currency in which the profits and losses are realized.

Requires at least the `General API - Read Only` permission in the API key settings.

- https://docs.kraken.com/api/docs/futures-api/trading/get-pnl-currency-preference

Returns
The current NPL preferences

Return type
dict

Listing 26: Futures Market: Get the users profit/loss preferences

```
>>> from kraken.futures import Market
>>> market = Market(key="api-key", secret="secret-key")
>>> market.get_pnl_preference()
{'result': 'success', 'serverTime': '2023-04-04T15:21:29.413Z', 'preferences':␣
↪[]}
```

get_public_execution_events(*tradeable: str*, *before: int | None = None*, *continuation_token: str | None = None*, *since: int | None = None*, *sort: str | None = None*, ***, *extra_params: dict | None = None*) → dict

Retrieve information about the public execution events. The returned `continuation_token` can be used to request more data.

- https://docs.kraken.com/api/docs/futures-api/history/get-public-execution-events

Parameters

- **tradeable** (*str*) – The contract to filter for
- **before** (*int | None, optional*) – Filter by time
- **continuation_token** (*str | None, optional*) – Token that can be used to continue requesting historical events
- **since** (*int | None, optional*) – Filter by a specifying a start point
- **sort** (*str | None, optional*) – Sort the results

Returns
The public execution events

Return type
dict

Listing 27: Futures Market: Get the public execution events

```
>>> from kraken.futures import Market
>>> Market().get_public_execution_events(tradeable="PI_XBTUSD")
{
    'elements': [
        {
            'uid': '9c74d4ba-a658-4208-891c-eee6e13bf910',
            'timestamp': 1680874894684,
            'event': {
                'Execution': {
                    'execution': {
                        'uid': '3df5cb59-d410-48f7-9c6f-ee9b849b9c91',
                        'makerOrder': {
                            'uid': 'a0d28216-54f8-4af0-9adc-0d0d4738936d',
                            'tradeable': 'PI_XBTUSD',
                            'direction': 'Buy',
                            'quantity': '626',
                            'timestamp': 1680874894675,
```

(continues on next page)

(continued from previous page)

```
                            'limitPrice': '27909.5',
                            'orderType': 'Post',
                            'reduceOnly': False,
                            'lastUpdateTimestamp': 1680874894675
                        },
                        'takerOrder': {
                            'uid': '09246639-9130-42fb-8d90-4ed39913456f',
                            'tradeable': 'PI_XBTUSD',
                            'direction': 'Sell',
                            'quantity': '626',
                            'timestamp': 1680874894684,
                            'limitPrice': '27909.50000000000',
                            'orderType': 'IoC',
                            'reduceOnly': False,
                            'lastUpdateTimestamp': 1680874894684
                        },
                        'timestamp': 1680874894684,
                        'quantity': '626',
                        'price': '27909.5',
                        'markPrice': '27915.01610466227',
                        'limitFilled': True,
                        'usdValue': '626.00'
                    },
                    'takerReducedQuantity': ''
                }
            }
        }, ...
    ],
    'len': 1000,
    'continuationToken': 'MTY4MDg2Nzg2ODkxOS85MDY0OTcwMTAxNA=='
}
```

get_public_mark_price_events(*tradeable: str*, *before: int | None = None*, *continuation_token: str | None = None*, *since: int | None = None*, *sort: str | None = None*, ***, *extra_params: dict | None = None*) → dict

Retrieve information about public mark price events. The returned `continuation_token` can be used to request more data.

- https://docs.kraken.com/api/docs/futures-api/history/get-public-price-events

Parameters

- **tradeable** (*str*) – The contract to filter for
- **before** (*int | None, optional*) – Filter by time
- **continuation_token** (*str | None, optional*) – Token that can be used to continue requesting historical events
- **since** (*int | None, optional*) – Filter by a specifying a start point
- **sort** (*str | None, optional*) – Sort the results

Returns

The public order events

Return type
dict

Listing 28: Futures Market: Get the public mark price events

```
>>> from kraken.futures import Market
>>> Market().get_public_mark_price_events(tradeable="PI_XBTUSD")
{
    'elements': [
        {
            'uid': '',
            'timestamp': 1680875273372,
            'event': {
                'MarkPriceChanged': {
                    'price': '27900.67795901584'
                }
            }
        }, {
            'uid': '',
            'timestamp': 1680875272263,
            'event': {
                'MarkPriceChanged': {
                    'price': '27900.09023205142'
                }
            }
        }, ...
    ],
    'len': 1000,
    'continuationToken': 'MTY4MDg3NDEyNzg3OC85MDY2MjI3ODIzMA=='
}
```

get_public_order_events(*tradeable: str*, *before: int | None = None*, *continuation_token: str | None = None*, *since: int | None = None*, *sort: str | None = None*, ***, *extra_params: dict | None = None*) → dict

Retrieve information about the public order events - filled, closed, opened, etc, for a specific contract.The returned `continuation_token` can be used to request more data.

- https://docs.kraken.com/api/docs/futures-api/history/get-public-order-events

Parameters

- **tradeable** (*str*) – The contract to filter for
- **before** (*int, optional*) – Filter by time
- **continuation_token** (*str, optional*) – Token that can be used to continue requesting historical events
- **since** (*int, optional*) – Filter by a specifying a start point
- **sort** (*str, optional*) – Sort the results

Returns
The public order events

Return type
dict

Listing 29: Futures Market: Get the public order events

```
>>> from kraken.futures import Market
>>> Market().get_public_order_events(tradeable="PI_XBTUSD")
{
    'elements': [
        {
            'uid': '430782d7-7b6d-472a-9e92-67047289d742',
            'timestamp': 1680875125649,
            'event': {
                'OrderPlaced': {
                    'order': {
                        'uid': 'f9aaf471-95ba-4fde-ab68-251f12f96e47',
                        'tradeable': 'PI_XBTUSD',
                        'direction': 'Sell',
                        'quantity': '652',
                        'timestamp': 1680875125649,
                        'limitPrice': '27927.5',
                        'orderType': 'Post',
                        'reduceOnly': False,
                        'lastUpdateTimestamp': 1680875125649
                    },
                    'reason': 'new_user_order',
                    'reducedQuantity': ''
                }
            }
        }, ...
    ],
    'len': 1000,
    'continuationToken': 'MTY4MDg3NTExMzc2OS85MDY2NDA1ODIyNw=='
}
```

get_resolutions(*tick_type: str*, *tradeable: str*, ***, *extra_params: dict | None = None*) → list[str]

Retrieve the list of available resolutions for a specific asset.

- https://docs.kraken.com/api/docs/futures-api/charts/resolutions

This function uses caching. Run `get_resolutions.cache_clear()` to clear.

Parameters

- **tick_type** (*str*) – The kind of data, based on `mark`, `spot`, or `trade`
- **tick_type** – The asset of interest

Returns

List of resolutions

Type

list[str]

Listing 30: Futures Market: Get the available resolutions

```
>>> from kraken.futures import Market
>>> Market().get_resolutions(tick_type="mark", tradeable="PI_XBTUSD")
['1h', '12h', '1w', '15m', '1d', '5m', '30m', '4h', '1m']
```

get_tick_types(*, *extra_params: dict | None = None*) → list[str]

Retrieve the available tick types that can be used for example to access the *kraken.futures.Market.get_ohlc()* endpoint.

- https://docs.kraken.com/api/docs/futures-api/charts/tick-types

This function uses caching. Run `get_tick_types.cache_clear()` to clear.

Returns
List of available tick types

Return type
list[str]

Listing 31: Futures Market: Get the available tick types

```
>>> from kraken.futures import Market
>>> Market().get_tick_types()
['mark', 'spot', 'trade']
```

get_tickers(*, *extra_params: dict | None = None*) → dict

Retrieve information about the current tickers of all futures contracts.

- https://docs.kraken.com/api/docs/futures-api/trading/get-tickers

Returns
The current tickers

Return type
dict

Listing 32: Futures Market: Get the available tickers

```
>>> from kraken.futures import Market
>>> Market().get_tickers()
{
    'tickers': [{
        'tag': 'perpetual',
        'pair': 'COMP:USD',
        'symbol': 'pf_compusd',
        'markPrice': 42.192,
        'bid': 42.14,
        'bidSize': 11.8,
        'ask': 42.244,
        'askSize': 80.7,
        'vol24h': 96.8,
        'volumeQuote': 4109.9678,
        'openInterest': 451.3,
        'open24h': 41.975,
        'indexPrice': 42.193,
        'last': 42.873,
        'lastTime': '2023-04-04T00:07:33.690Z',
        'lastSize': 13.9,
        'suspended': False,
        'fundingRate': 0.0002207140788888884,
        'fundingRatePrediction': 6.3914700437486e-05,
```

(continues on next page)

(continued from previous page)

```
        'postOnly': False
    }, ...]
}
```

get_trade_history(*symbol: str | None = None*, *lastTime: str | None = None*, ***, *extra_params: dict | None = None*) → dict

Retrieve the trade history (max 100 entries), can be filtered using the parameters.

- https://docs.kraken.com/api/docs/futures-api/trading/get-history

Parameters

- **symbol** (*str, optional*) – The asset to filter for
- **lastTime** (*str, optional*) – Filter by time

Returns
Trade history

Return type
dict

Listing 33: Futures Market: Get the public trade history

```
>>> from kraken.futures import Market
>>> Market().get_trade_history(symbol="PI_XBTUSD")
{
    'history': [{
            'time': '2023-04-04T05:28:27.926Z',
            'trade_id': 100,
            'price': 27913,
            'size': 2456,
            'side': 'sell',
            'type': 'fill',
            'uid': 'de7a2eca-f2bc-4afb-860d-522a9dcc0b12'
        }, {
            'time': '2023-04-04T05:28:28.795Z',
            'trade_id': 99, 'price': 27913.5,
            'size': 3000, 'side': 'sell',
            'type': 'fill',
            'uid': 'c985c7ea-30a5-4456-b857-a4ec9156fb3b'
        }, ...
    ]
```

get_tradeable_products(*tick_type: str*, ***, *extra_params: dict | None = None*) → list[str]

Retrieve a list containing the tradeable assets on the futures market.

- https://docs.kraken.com/api/docs/futures-api/charts/symbols

This function uses caching. Run `get_tradeable_products.cache_clear()` to clear.

Parameters
tick_type (*str*) – The kind of data, based on `mark`, `spot`, or `trade`

Returns
List of tradeable assets

Type
list[str]

Listing 34: Futures Market: Get the tradeable products

```
>>> from kraken.futures import Market
>>> Market().get_tradeable_products(tick_type="trade")
["PI_XBTUSD", "PF_XBTUSD", "PF_SOLUSD", ...]
```

request(*method: str*, *uri: str*, *post_params: dict | None = None*, *query_params: dict | None = None*, *timeout: int = 10*, ***, *auth: bool = True*, *return_raw: bool = False*, *extra_params: str | dict | None = None*) → dict | list | Response

Handles the requested requests, by sending the request, handling the response, and returning the message or in case of an error the respective Exception.

Parameters

- **method** (*str*) – The request method, e.g., `GET`, `POST`, and `PUT`
- **uri** (*str*) – The endpoint to send the message
- **post_params** (*dict, optional*) – The query parameter of the request (default: `None`)
- **extra_params** (*str | dict, optional*) – Additional query parameter of the request (default: `None`)
- **query_params** (*dict, optional*) – The query parameter of the request (default: `None`)
- **timeout** (*int*) – Timeout for the request (default: `10`)
- **auth** (*bool*) – If the request needs authentication (default: `True`)
- **return_raw** (*bool, optional*) – If the response should be returned without parsing. This is used for example when requesting an export of the trade history as .zip archive.

Raises
kraken.exceptions.* – If the response contains errors

Returns
The response

Return type
dict | list | requests.Response

set_leverage_preference(*symbol: str*, *maxLeverage: str | float | None = None*, ***, *extra_params: dict | None = None*) → dict

Set a new leverage preference for a specific futures contract.

Requires the `General API - Full Access` permission in the API key settings.

- https://docs.kraken.com/api/docs/futures-api/trading/set-leverage-setting

Parameters

- **symbol** (*str, optional*) – The symbol to set the preference
- **maxLeverage** (*str | float, optional*) – The maximum allowed leverage for a futures contract

Returns
Information about the success or fail

Return type
dict

Listing 35: Futures Market: Set the users leverage preferences

```
>>> from kraken.futures import Market
>>> market = Market(key="api-key", secret="secret-key")
>>> market.set_leverage_preference(symbol="PF_XBTUSD", maxLeverage=2)
{'result': 'success', 'serverTime': '2023-04-04T05:59:49.576Z'}
```

set_pnl_preference(*symbol: str*, *pnlPreference: str*, ***, *extra_params: dict | None = None*) → dict

Modify or set the current PNL preference of the user. This can be used to define a specific currency that should be used to realize profits and losses. The default is the quote currency of the futures contract.

Requires the `General API - Full Access` permission in the API key settings.

- https://docs.kraken.com/api/docs/futures-api/trading/set-pnl-currency-preference

Parameters

- **symbol** (*str*) – The asset pair or futures contract
- **pnlPreference** – The currency to pay out the profits and losses

Returns
Success or failure of the request

Return type
dict

Listing 36: Futures Market: Set the users profit/loss preferences

```
>>> from kraken.futures import Market
>>> market = Market(key="api-key", secret="secret-key")
>>> market.set_pnl_preference(symbol="PF_XBTUSD", pnlPreference="USD")
{'result': 'success', 'serverTime': '2023-04-04T15:24:18.406Z'}
```

class kraken.futures.**Trade**(*key: str = ''*, *secret: str = ''*, *url: str = ''*, *proxy: str | None = None*, ***, *sandbox: bool = False*)

Bases: *FuturesClient*

Class that implements the Kraken Futures trade client

If the sandbox environment is chosen, the keys must be generated from here: https://demo-futures.kraken.com/settings/api

Parameters

- **key** (*str, optional*) – Futures API public key (default: "")
- **secret** (*str, optional*) – Futures API secret key (default: "")
- **url** (*str, optional*) – Alternative URL to access the Futures Kraken API (default: https://futures.kraken.com)
- **proxy** (*str, optional*) – proxy URL, may contain authentication information
- **sandbox** (*bool, optional*) – If set to `True` the URL will be https://demo-futures.kraken.com (default: `False`)

Listing 37: Futures Trade: Create the trade client

```
>>> from kraken.futures import Trade
>>> trade = Trade() # unauthenticated
>>> trade = Trade(key="api-key", secret="secret-key") # authenticated
```

Listing 38: Futures Trade: Create the trade client as context manager

```
>>> from kraken.futures import Trade
>>> with Trade(key="api-key", secret="secret-key") as trade:
...     print(trade.get_fills())
```

cancel_all_orders(*symbol: str | None = None*, *, *extra_params: dict | None = None*) → dict

Cancels all open orders, can be filtered by symbol.

Requires the `General API - Full Access` permission in the API key settings.

- https://docs.kraken.com/api/docs/futures-api/trading/cancel-all-orders

Parameters
symbol (*str, optional*) – Filter by symbol

Returns
Information about the success or failure

Return type
dict

Listing 39: Futures Trade: Cancel all open orders

```
>>> from kraken.futures import Trade
>>> trade = Trade(key="api-key", secret="secret-key")
>>> trade.cancel_all_orders()
{
    'result': 'success',
    'cancelStatus': {
        'receivedTime': '2023-04-04T17:09:09.986Z',
        'cancelOnly': 'all',
        'status': 'cancelled',
        'cancelledOrders': [
            {
                'order_id': 'fc589be9-5095-48f0-b6f1-a2dfad6d9677'
            }, {
                'order_id': '0365942e-4850-4e41-90c3-a10f96f7baaf'
            }
        ],
        'orderEvents': []
    },
    'serverTime': '2023-04-04T17:09:09.987Z'
}
```

cancel_order(*order_id: str | None = None*, *cliOrdId: str | None = None*, *processBefore: str | None = None*, *, *extra_params: dict | None = None*) → dict

This endpoint can be used to cancel a specific order by `order_id` or `cliOrdId`.

Requires the `General API - Full Access` permission in the API key settings.

- https://docs.kraken.com/api/docs/futures-api/trading/cancel-order

Parameters

- **order_id** (*str, optional*) – The order_id to cancel
- **cliOrdId** (*str, optional*) – The client defined order id
- **processBefore** (*str, optional*) – Process before timestamp otherwise reject

Raises

ValueError – If both `order_id` and `cliOrdId` are not set

Returns

Success or failure

Return type

dict

Listing 40: Futures Trade: Cancel an order

```
>>> from kraken.futures import Trade
>>> trade = Trade(key="api-key", secret="secret-key")
>>> trade.cancel_order(order_id="fc589be9-5095-48f0-b6f1-a2dfad6d9677")
{
    'result': 'success',
    'cancelStatus': {
        'status': 'notFound',
        'receivedTime': '2023-04-04T17:18:11.628Z'
    },
    'serverTime': '2023-04-04T17:18:11.628Z'
}
```

create_batch_order(*batchorder_list: list[dict]*, *processBefore: str | None = None*, ***, *extra_params: dict | None = None*) → dict

Create multiple orders at once using the batch order endpoint.

Requires the `General API - Full Access` permission in the API key settings.

- https://docs.kraken.com/api/docs/futures-api/trading/send-batch-order

Parameters

- **batchorder_list** (*list[dict]*) – List of order instructions (see example below - or the linked official Kraken documentation)
- **processBefore** (*str, optional*) – Process before timestamp otherwise reject

Returns

Information about the submitted request

Return type

dict

Listing 41: Futures Trade: Create a batch order

```
>>> from kraken.futures import Trade
>>> trade = Trade(key="api-key", secret="secret-key")
>>> trade.create_batch_order(
...     batchorder_list=[
...         {
...             "order": "send",
...             "order_tag": "1",
...             "orderType": "lmt",
...             "symbol": "PI_XBTUSD",
...             "side": "buy",
...             "size": 5,
...             "limitPrice": 1.00,
...             "cliOrdId": "my_another_client_id",
...         },
...         {
...             "order": "send",
...             "order_tag": "2",
...             "orderType": "stp",
...             "symbol": "PI_XBTUSD",
...             "side": "buy",
...             "size": 1,
...             "limitPrice": 2.00,
...             "stopPrice": 3.00,
...         },
...         {
...             "order": "cancel",
...             "order_id": "e35d61dd-8a30-4d5f-a574-b5593ef0c050",
...         },
...         {
...             "order": "cancel",
...             "cliOrdId": "my_client_id",
...         },
...     ],
... )
{
    'result': 'success',
    'serverTime': '2023-04-04T17:03:36.100Z',
    'batchStatus': [
        {
            'status': 'insufficientAvailableFunds',
            'order_tag': '1',
            'orderEvents': []
        }, {
            'status': 'placed',
            'order_tag': '2',
            'order_id':
            'fc589be9-5095-48f0-b6f1-a2dfad6d9677',
            'dateTimeReceived': '2023-04-04T17:03:36.053Z',
            'orderEvents': [
                {
                    'orderTrigger': {
```

(continues on next page)

(continued from previous page)

```
                        'uid': 'fc589be9-5095-48f0-b6f1-a2dfad6d9677',
                        'clientId': None,
                        'type': 'lmt',
                        'symbol': 'pi_xbtusd',
                        'side': 'buy',
                        'quantity': 1,
                        'limitPrice': 2.0,
                        'triggerPrice': 3.0,
                        'triggerSide': 'trigger_above',
                        'triggerSignal':
                        'last_price',
                        'reduceOnly': False,
                        'timestamp': '2023-04-04T17:03:36.053Z',
                        'lastUpdateTimestamp': '2023-04-04T17:03:36.053Z',
                        'startTime': None
                    },
                    'type': 'PLACE'
                }
            ]
        }, {
            'status': 'notFound',
            'order_id': 'e35d61dd-8a30-4d5f-a574-b5593ef0c050',
            'orderEvents': []
        }, {
            'status': 'notFound',
            'cliOrdId': 'my_client_id',
            'orderEvents': []
        }
    ]
}
```

create_order(*orderType: str*, *size: str | float*, *symbol: str*, *side: str*, *cliOrdId: str | None = None*, *limitPrice: str | float | None = None*, *reduceOnly: bool | None = None*, *stopPrice: str | float | None = None*, *triggerSignal: str | None = None*, *trailingStopDeviationUnit: str | None = None*, *trailingStopMaxDeviation: str | None = None*, *processBefore: str | None = None*, ***, *extra_params: dict | None = None*) → dict

Create and place an order on the futures market.

Requires the `General API - Full Access` permission in the API key settings.

- https://docs.kraken.com/api/docs/futures-api/trading/send-order

Parameters

- **orderType** (*str*) – The order type, one of `lmt`, `post`, `ioc`, `mkt`, `stp`, `take_profit`, `trailing_stop` (https://support.kraken.com/hc/en-us/sections/200577136-Order-types)
- **size** (*str | float*) – The volume of the position
- **symbol** (*str*) – The symbol to trade
- **side** (*str*) – Long or Short, i.e.,: `buy` or `sell`
- **cliOrdId** (*str, optional*) – A user defined order id
- **limitPrice** (*str | float, optional*) – Define a custom limit price

- **reduceOnly** (*bool, optional*) – Reduces existing positions if set to `True`
- **stopPrice** (*str, optional*) – Define a price when to exit the order. Required for specific order types
- **triggerSignal** (*str, optional*) – Define a trigger for specific orders (must be one of `mark`, `index`, `last`)
- **trailingStopDeviationUnit** (*str, optional*) – See referenced Kraken documentation
- **trailingStopMaxDeviation** (*str, optional*) – See referenced Kraken documentation
- **processBefore** (*str, optional*) – Process before timestamp otherwise reject

Returns
Success or failure

Return type
dict

Listing 42: Futures Trade: Create and submit a new market order

```
>>> trade.create_order(
...     orderType="mkt",
...     size=5,
...     side="buy",
...     symbol="PI_ETHUSD",
... )
{
    'result': 'success',
    'sendStatus': {
        'order_id': '67d3a732-b0d3-49e7-9577-45b31bceb833',
        'status': 'placed',
        'receivedTime': '2023-04-08T11:59:23.887Z',
        'orderEvents': [
            {
                'executionId': '495aae73-7cf7-4dfc-8963-3b766d8150de',
                'price': 1869.175,
                'amount': 5,
                'orderPriorEdit': None,
                'orderPriorExecution': {
                    'orderId': '67d3a732-b0d3-49e7-9577-45b31bceb833',
                    'cliOrdId': None,
                    'type': 'ioc',
                    'symbol': 'pi_ethusd',
                    'side': 'buy',
                    'quantity': 5,
                    'filled': 0,
                    'limitPrice': 1887.85,
                    'reduceOnly': False,
                    'timestamp': '2023-04-08T11:59:23.887Z',
                    'lastUpdateTimestamp': '2023-04-08T11:59:23.887Z'
                },
                'takerReducedQuantity': None,
```

(continues on next page)

(continued from previous page)

```
                'type': 'EXECUTION'
            }
        ]
    },
    'serverTime': '2023-04-08T11:59:23.888Z'
}
```

Listing 43: Futures Trade: Create and submit a new limit order

```
>>> from kraken.futures import Trade
>>> trade = Trade(key="api-key", secret="secret-key")
>>> trade.create_order(
...     orderType="lmt",
...     size=1000,
...     symbol="PF_ETHUSD",
...     side="buy",
...     limitPrice=1200.0,
... )
{
    'result': 'success',
    'sendStatus': {
        'order_id': '2ce038ae-c144-4de7-a0f1-82f7f4fca864',
        'status': 'placed',
        'receivedTime': '2023-04-07T15:18:04.699Z',
        'orderEvents': [
            {
                'order': {
                    'orderId': '2ce038ae-c144-4de7-a0f1-82f7f4fca864',
                    'cliOrdId': None,
                    'type': 'lmt',
                    'symbol': 'pi_ethusd',
                    'side': 'buy',
                    'quantity': 100,
                    'filled': 0,
                    'limitPrice': 1200.0,
                    'reduceOnly': False,
                    'timestamp': '2023-04-07T15:18:04.699Z',
                    'lastUpdateTimestamp': '2023-04-07T15:18:04.699Z'
                },
                'reducedQuantity': None,
                'type': 'PLACE'
            }
        ]
    },
    'serverTime': '2023-04-07T15:18:04.700Z'
}
```

Listing 44: Futures Trade: Create and submit a new take profit order

```
>>> trade.create_order(
...     orderType="take_profit",
...     size=10,
```

(continues on next page)

(continued from previous page)

```
...     side="buy",
...     symbol="PI_ETHUSD",
...     limitPrice=2500.0,
...     triggerSignal="last",
...     stopPrice=2498.4,
... )
{
    'result': 'success',
    'sendStatus': {
        'order_id': 'e58ed100-1fb8-4e6c-a5ea-1cf85b0f0654',
        'status': 'placed',
        'receivedTime': '2023-04-07T15:12:08.131Z',
        'orderEvents': [
            {
                'orderTrigger': {
                    'uid': 'e58ed100-1fb8-4e6c-a5ea-1cf85b0f0654',
                    'clientId': None,
                    'type': 'lmt',
                    'symbol': 'pi_ethusd',
                    'side': 'buy',
                    'quantity': 10,
                    'limitPrice': 1860.0,
                    'triggerPrice': 1880.4,
                    'triggerSide': 'trigger_below',
                    'triggerSignal': 'last_price',
                    'reduceOnly': False,
                    'timestamp': '2023-04-07T15:12:08.131Z',
                    'lastUpdateTimestamp': '2023-04-07T15:12:08.131Z',
                    'startTime': None
                },
                'type': 'PLACE'
            }
        ]
    },
    'serverTime': '2023-04-07T15:12:08.131Z'
}
```

dead_mans_switch(*timeout: int | None = 0*, **, extra_params: dict | None = None*) → dict

The Death Man's Switch can be used to cancel all orders after a specific timeout. If the timeout is set to 60, all orders will be cancelled after 60 seconds. The timeout can be pushed back by accessing this endpoint over and over again. Set the timeout to zero to reset.

Requires the `General API - Full Access` permission in the API key settings.

- https://docs.kraken.com/api/docs/futures-api/trading/cancel-all-orders-after

Parameters
timeout (*int, optional*) – The timeout in seconds

Returns
Success or failure

Return type
dict

Listing 45: Futures Trade: Setup the Death man's Switch

```
>>> from kraken.futures import Trade
>>> trade = Trade(key="api-key", secret="secret-key")
>>> trade.dead_mans_switch(timeout=60)
{
    'result': 'success',
    'serverTime': '2023-04-04T17:14:34.113Z',
    'status': {
        'currentTime': '2023-04-04T17:14:34.076Z',
        'triggerTime': '0'
    }
}
```

edit_order(*orderId: str | None = None*, *cliOrdId: str | None = None*, *limitPrice: str | float | None = None*, *size: str | float | None = None*, *stopPrice: str | float | None = None*, *processBefore: str | None = None*, ***, *extra_params: dict | None = None*) → dict

Edit an open order.

Requires the `General API - Full Access` permission in the API key settings.

- https://docs.kraken.com/api/docs/futures-api/trading/edit-order-spring

Parameters

- **orderId** (*str, optional*) – The order id to cancel
- **cliOrdId** (*str, optional*) – The client defined order id
- **limitPrice** (*str | float None*) – The new limit price
- **size** (*str | float, optional*) – The new size of the position
- **stopPrice** (*str | float, optional*) – The stop price
- **processBefore** (*str, optional*) – Process before timestamp otherwise reject

Raises

ValueError – If both `orderId` and `cliOrdId` are not set

Returns

Success or failure

Return type

dict

Listing 46: Futures Trade: Edit an open order

```
>>> from kraken.futures import Trade
>>> trade = Trade(key="api-key", secret="secret-key")
>>> trade.edit_order(orderId="fc589be9-5095-48f0-b6f1-a2dfad6d9677", size=100)
{
    'result': 'success',
    'serverTime': '2023-04-04T17:24:53.233Z',
    'editStatus': {
        'status': 'orderForEditNotFound',
        'orderId': 'fc589be9-5095-48f0-b6f1-a2dfad6d9677',
```

(continues on next page)

(continued from previous page)

```
        'receivedTime':
        '2023-04-04T17:24:53.233Z',
        'orderEvents': []
    }
}
```

get_fills(*lastFillTime: str | None = None*, *, *extra_params: dict | None = None*) → dict

Return the current fills of the user.

Requires at least the `General API - Read Only` permission in the API key settings.

- https://docs.kraken.com/api/docs/futures-api/trading/get-fills

Parameters
lastFillTime (*str, optional*) – Filter by last filled timestamp

Returns
Fills

Return type
dict

Listing 47: Futures Trade: Get the recent fills

```
>>> from kraken.futures import Trade
>>> trade = Trade(key="api-key", secret="secret-key")
>>> trade.get_fills()
{
    'result': 'success',
    'fills': [{
        'fill_id': '15dae264-01e9-4d4c-8962-2f49b98c46f6',
        'symbol': 'pi_ethusd',
        'side': 'buy',
        'order_id': '267372ec-272f-4ca7-9b8c-99a0dc8f781c',
        'size': 5,
        'price': 1859.075,
        'fillTime': '2023-04-07T15:07:46.540Z',
        'fillType': 'taker'
    }, ...],
    'serverTime': '2023-04-07T15:23:48.705Z'
}
```

get_nonce() → str

Return a new nonce

get_orders_status(*orderIds: str | list[str] | None = None*, *cliOrdIds: str | list[str] | None = None*, *, *extra_params: dict | None = None*) → dict

Get the status of multiple orders.

Requires at least the `General API - Read Only` permission in the API key settings.

- https://docs.kraken.com/api/docs/futures-api/trading/get-order-status

Parameters

- **orderIds** (*str | list[str], optional*) – The order ids to cancel

- **cliOrdId** (*str | list[str], optional*) – The client defined order ids

Returns
Success or failure

Return type
dict

Listing 48: Futures Trade: Get the order status

```
>>> from kraken.futures import Trade
>>> trade = Trade(key="api-key", secret="secret-key")
>>> trade.get_orders_status(
...     orderIds=[
...         "2c611222-bfe6-42d1-9f55-77bddc01a313",
...         "5f204f95-4354-4610-bb3b-c902ad333012"
... ])
{'result': 'success', 'serverTime': '2023-04-04T17:27:29.667Z', 'orders': []}
```

request(*method: str*, *uri: str*, *post_params: dict | None = None*, *query_params: dict | None = None*, *timeout: int = 10*, ***, *auth: bool = True*, *return_raw: bool = False*, *extra_params: str | dict | None = None*) → dict | list | Response

Handles the requested requests, by sending the request, handling the response, and returning the message or in case of an error the respective Exception.

Parameters

- **method** (*str*) – The request method, e.g., `GET`, `POST`, and `PUT`
- **uri** (*str*) – The endpoint to send the message
- **post_params** (*dict, optional*) – The query parameter of the request (default: `None`)
- **extra_params** (*str | dict, optional*) – Additional query parameter of the request (default: `None`)
- **query_params** (*dict, optional*) – The query parameter of the request (default: `None`)
- **timeout** (*int*) – Timeout for the request (default: `10`)
- **auth** (*bool*) – If the request needs authentication (default: `True`)
- **return_raw** (*bool, optional*) – If the response should be returned without parsing. This is used for example when requesting an export of the trade history as .zip archive.

Raises
kraken.exceptions.* – If the response contains errors

Returns
The response

Return type
dict | list | requests.Response

class kraken.futures.**Funding**(*key: str = ''*, *secret: str = ''*, *url: str = ''*, *proxy: str | None = None*, ***, *sandbox: bool = False*)

Bases: *FuturesClient*

Class that implements the Kraken Futures Funding client

If the sandbox environment is chosen, the keys must be generated from here: https://demo-futures.kraken.com/settings/api

Parameters

- **key** (*str, optional*) – Futures API public key (default: "")
- **secret** (*str, optional*) – Futures API secret key (default: "")
- **url** (*str, optional*) – Alternative URL to access the Futures Kraken API (default: https://futures.kraken.com)
- **proxy** (*str, optional*) – proxy URL, may contain authentication information
- **sandbox** (*bool, optional*) – If set to `True` the URL will be https://demo-futures.kraken.com (default: `False`)

Listing 49: Futures Funding: Create the funding client

```
>>> from kraken.futures import Funding
>>> funding = Funding() # unauthenticated
>>> funding = Funding(key="api-key", secret="secret-key") # authenticated
```

Listing 50: Futures Funding: Create the funding client as context manager

```
>>> from kraken.futures import Funding
>>> with Funding(key="api-key", secret="secret-key") as funding:
...     print(funding.get_historical_funding_rates(symbol="PI_XBTUSD"))
```

get_historical_funding_rates(*symbol: str*, *, *extra_params: dict | None = None*) → dict

Retrieve information about the historical funding rates of a specific `symbol`

- https://docs.kraken.com/api/docs/futures-api/trading/historical-funding-rates

Parameters
symbol (*str*) – The futures symbol to filter for

Returns
The funding rates for a specific asset/contract

Return type
dict

Listing 51: Futures Funding: Get the historical funding rates

```
>>> from kraken.futures import Funding
>>> Funding().get_historical_funding_rates(symbol="PI_XBTUSD")
{
    'rates': [
        {
            'timestamp': '2019-02-27T16:00:00.000Z',
            'fundingRate': 1.31656208775e-07,
            'relativeFundingRate': 0.0005
        }, {
            'timestamp': '2019-02-27T20:00:00.000Z',
            'fundingRate': 1.30695377827e-07,
            'relativeFundingRate': 0.0005
        }, ...
```

(continues on next page)

(continued from previous page)

```
    ]
}
```

get_nonce() → str

Return a new nonce

initiate_subaccount_transfer(*amount: str | float*, *fromAccount: str*, *fromUser: str*, *toAccount: str*, *toUser: str*, *unit: str*, ***, *extra_params: dict | None = None*) → dict

Submit a request to transfer funds between the regular and subaccount.

Requires the `General API - Full Access` and `Withdrawal API - Full access` permissions in the API key settings.

- https://docs.kraken.com/api/docs/futures-api/trading/sub-account-transfer

Parameters

- **amount** (*str | float*) – The volume to transfer
- **fromAccount** (*str*) – The account to withdraw from
- **fromUser** (*str*) – The user account to transfer from
- **toAccount** (*str*) – The account to deposit to
- **unit** (*str*) – The asset to transfer

Listing 52: Futures Funding: Transfer funds between subaccounts

```
>>> from kraken.futures import Funding
>>> funding = Funding(key="api-key", secret="secret-key")
>>> funding.initiate_subaccount_transfer(
...     amount='2',
...     fromAccount='MyCashWallet',
...     fromUser='Subaccount1',
...     toAccount='MyCashWallet',
...     toUser='Subaccount2',
...     unit='XBT'
... ))
```

initiate_wallet_transfer(*amount: str | float*, *fromAccount: str*, *toAccount: str*, *unit: str*, ***, *extra_params: dict | None = None*) → dict

Submit a wallet transfer request to transfer funds between margin accounts.

Requires the `General API - Full Access` and `Withdrawal API - Full access` permissions in the API key settings.

- https://docs.kraken.com/api/docs/futures-api/trading/transfer

Parameters

- **amount** (*str | float*) – The volume to transfer
- **fromAccount** (*str*) – The account to withdraw from
- **fromAccount** – The account to deposit to
- **unit** (*str*) – The currency or asset to transfer

Listing 53: Futures Funding: Transfer funds between wallets

```
>>> from kraken.futures import Funding
>>> funding = Funding(key="api-key", secret="secret-key")
>>> funding.initiate_wallet_transfer(
...     amount='100',
...     fromAccount='some cash or margin account',
...     toAccount='another cash or margin account',
...     unit='ADA'
... ))
{
    'result': 'success',
    'serverTime': '2023-04-07T15:23:45.196Z"
}
```

initiate_withdrawal_to_spot_wallet(*amount: str | float*, *currency: str*, *sourceWallet: str | None = None*, ***, *extra_params: dict | None = None*) → dict

Enables the transfer of funds between the futures and spot wallet.

Requires the `General API - Full Access` and `Withdrawal API - Full access` permissions in the API key settings.

- https://docs.kraken.com/api/docs/futures-api/trading/withdrawal

Parameters

- **amount** (*str | float*) – The volume to transfer
- **currency** (*str*) – The asset or currency to transfer
- **sourceWallet** (*str, optional*) – The wallet to withdraw from (default: `cash`)

Raises

ValueError – If this function is called within the sandbox/demo environment

Listing 54: Futures Funding: Transfer funds between Spot and Futures wallets

```
>>> from kraken.futures import Funding
>>> funding = Funding(key="api-key", secret="secret-key")
>>> funding.initiate_withdrawal_to_spot_wallet(
...     amount=100,
...     currency='USDT',
...     sourceWallet='cash'
... ))
```

request(*method: str*, *uri: str*, *post_params: dict | None = None*, *query_params: dict | None = None*, *timeout: int = 10*, ***, *auth: bool = True*, *return_raw: bool = False*, *extra_params: str | dict | None = None*) → dict | list | Response

Handles the requested requests, by sending the request, handling the response, and returning the message or in case of an error the respective Exception.

Parameters

- **method** (*str*) – The request method, e.g., `GET`, `POST`, and `PUT`
- **uri** (*str*) – The endpoint to send the message

- **post_params** (*dict, optional*) – The query parameter of the request (default: None)
- **extra_params** (*str | dict, optional*) – Additional query parameter of the request (default: None)
- **query_params** (*dict, optional*) – The query parameter of the request (default: None)
- **timeout** (*int*) – Timeout for the request (default: 10)
- **auth** (*bool*) – If the request needs authentication (default: True)
- **return_raw** (*bool, optional*) – If the response should be returned without parsing. This is used for example when requesting an export of the trade history as .zip archive.

Raises
kraken.exceptions.* – If the response contains errors

Returns
The response

Return type
dict | list | requests.Response

CHAPTER

NINE

FUTURES WEBSOCKETS

class kraken.futures.**FuturesWSClient**(*key: str = ''*, *secret: str = ''*, *url: str = ''*, *callback: Callable | None = None*, ***, *sandbox: bool = False*)

Bases: *FuturesAsyncClient*

Class to access public and (optional) private/authenticated websocket connection.

So far there are no trade endpoints that can be accessed using the Futures Kraken API. If this has changed and is not implemented yet, please open an issue at https://github.com/btschwertfeger/python-kraken-sdk/issues

- https://docs.kraken.com/api/docs/guides/futures-websockets

Parameters

- **key** (*str, optional*) – The Kraken Futures API key to access private endpoints
- **secret** (*str, optional*) – The Kraken Futures Secret key to access private endpoints
- **url** (*str, optional*) – Set a custom URL (default: `futures.kraken.com/ws/v1`)
- **sandbox** (*bool, optional*) – Use the Kraken Futures demo environment (URL will switch to `demo-futures.kraken.com/ws/v1`, default: `False`)

Listing 1: Futures Websocket: Create the websocket client

```
import asyncio
from kraken.futures import FuturesWSClient

# Create the custom client
class Client(FuturesWSClient):
    async def on_message(self, event: dict) -> None:
        print(event)

async def main() -> None:
    client = Client()     # unauthenticated
    auth_client = Client( # authenticated
        key="api-key",
        secret="secret-key"
    )

    # open the websocket connections
    await client.start()
    await auth_client.start()

```

(continues on next page)

(continued from previous page)

```
    # now you can subscribe to channels using
    await client.subscribe(
        feed='ticker',
        products=["XBTUSD", "DOT/EUR"]
    )
    # the messages can be used within the `on_message` callback method

    while True:
        await asyncio.sleep(6)

if __name__ == "__main__":
    try:
        asyncio.run(main())
    except KeyboardInterrupt:
        pass
```

Listing 2: Futures Websocket: Create the websocket client as context manager

```
import asyncio
from kraken.futures import FuturesWSClient

async def on_message(message):
    print(message)

async def main() -> None:
    async with FuturesWSClient(callback=on_message) as session:
        await session.subscribe(feed="ticker", products=["PF_XBTUSD"])

    while True:
        await asyncio.sleep(6)

if __name__ == "__main__":
    try:
        asyncio.run(main())
    except KeyboardInterrupt:
        pass
```

async_close() → None

Closes the aiohttp session

async close() → None

Method to stop the websocket connection.

property exception_occur: bool

Returns True if the connection was stopped due to an exception.

get_active_subscriptions() → list[dict]

Returns the list of active subscriptions.

Returns

List of active subscriptions including the feed names, products and additional information.

Return type

list[dict]

Initialize your client as described in *kraken.futures.FuturesWSClient* to run the following example:

Listing 3: Futures Websocket: Get the active subscriptions

```
>>> from kraken.futures import FuturesWSClient
...
>>> FuturesWSClient.get_active_subscriptions()
[
    {
        "event": "subscribe",
        "feed": "ticker,
        "product_ids": ["PI_XBTUSD"]
    }, {
        "event": "subscribe",
        "feed": "open_orders,
    }, ...
]
```

static get_available_private_subscription_feeds() → list[str]

Return all available private feeds that can be un-/subscribed to/from using the Kraken Futures API

Returns
List of available private feeds

Return type
list[str]

Listing 4: Futures Websocket: Get the available private subscription feeds

```
>>> from kraken.futures import FuturesWSClient
>>> FuturesWSClient.get_available_private_subscription_feeds()
[
    "fills", "open_positions", "open_orders",
    "open_orders_verbose", "balances",
    "deposits_withdrawals", "account_balances_and_margins",
    "account_log", "notifications_auth"
]
```

static get_available_public_subscription_feeds() → list[str]

Return all available public feeds that can be un-/subscribed using the Kraken Futures API.

Returns
List of available public feeds

Return type
list[str]

Listing 5: Futures Websocket: Get the available public subscription feeds

```
>>> from kraken.futures import FuturesWSClient
>>> FuturesWSClient.get_available_private_subscription_feeds()
[
    "trade", "book", "ticker",
```

(continues on next page)

(continued from previous page)

```
    "ticker_lite", "heartbeat"
]
```

get_nonce() → str

Return a new nonce

get_sign_challenge(*challenge: str*) → str

Sign the challenge/message using the secret key

Parameters
challenge (*str*) – The challenge/message to sign

Returns
The signed message

Raises
kraken.exceptions.KrakenAuthenticationError – If the credentials are not valid

Return type
str

property is_auth: bool

Checks if key and secret are set.

Returns
`True` if the credentials are set, else `False`

Return type
bool

Listing 6: Futures Websocket: Check if the credentials are set

```
>>> from kraken.futures import FuturesWSClient
>>> FuturesWSClient().is_auth()
False
```

property key: str

Returns the API key

async on_message(*message: dict*) → None

Method that serves as the default callback function Calls the defined callback function (if defined) or overload this function.

This is the default method which just logs the messages. In production you want to overload this with your custom methods, as shown in the Example of *kraken.futures.FuturesWSClient*.

Parameters
message (*dict*) – The message that was send by Kraken via the websocket connection.

Return type
None

async request(*method: str*, *uri: str*, *post_params: dict | None = None*, *query_params: dict | None = None*, *timeout: int = 10*, ***, *auth: bool = True*, *return_raw: bool = False*) → dict | list | aiohttp.ClientResponse | Awaitable

Handles the requested requests, by sending the request, handling the response, and returning the message or in case of an error the respective Exception.

Parameters

- **method** (*str*) – The request method, e.g., GET, POST, and PUT
- **uri** (*str*) – The endpoint to send the message
- **post_params** (*dict, optional*) – The query parameter of the request (default: None)
- **extra_params** (*str | dict, optional*) – Additional query parameter of the request (default: None)
- **query_params** (*dict, optional*) – The query parameter of the request (default: None)
- **timeout** (*int*) – Timeout for the request (default: 10)
- **auth** (*bool*) – If the request needs authentication (default: True)
- **return_raw** (*bool, optional*) – If the response should be returned without parsing. This is used for example when requesting an export of the trade history as .zip archive.

Raises
kraken.exceptions.* – If the response contains errors

Returns
The response

Return type
dict | list | requests.Response

async start() → None

Method to start the websocket connection.

stop() → None

Method to stop the websocket connection.

async subscribe(*feed: str*, *products: list[str] | None = None*) → None

Subscribe to a Futures websocket channel/feed. For some feeds authentication is required.

- https://docs.kraken.com/api/docs/guides/futures-websockets

Parameters

- **feed** (*str*) – The websocket feed/channel to subscribe to
- **products** (*list[str], optional*) – The products/futures contracts to subscribe to

Raises
TypeError – If the parameters don't match the requirements set by the Kraken API

Initialize your client as described in *kraken.futures.FuturesWSClient* to run the following example:

Listing 7: Futures Websocket: Subscribe to a feed

```
>>> await bot.subscribe(feed='ticker', products=["XBTUSD", "DOT/EUR"])
```

Success or failures are sent over the websocket connection and can be received via the default *kraken.futures.FuturesWSClient.on_message()* or a custom callback function.

async unsubscribe(*feed: str*, *products: list[str] | None = None*) → None

Subscribe to a Futures websocket channel/feed. For some feeds authentication is required.

- https://docs.kraken.com/api/docs/guides/futures-websockets

Parameters

- **feed** (*str*) – The websocket feed/channel to unsubscribe from
- **products** (*list[str], optional*) – The products/futures contracts to unsubscribe from

Raises

TypeError – If the parameters don't match the requirements set by the Kraken API

Initialize your client as described in *kraken.futures.FuturesWSClient* to run the following example:

Listing 8: Futures Websocket: Unsubscribe from a feed

```
>>> await bot.unsubscribe(feed='ticker', products=["XBTUSD", "DOT/EUR"])
```

Success or failures are sent over the websocket connection and can be received via the default `kraken.futures.FuturesWSClient.on_message`() or a custom callback function.

CHAPTER

TEN

CUSTOM EXCEPTIONS

The python-kraken-sdk provides lots of custom exceptions which are listed below. Module that provides custom exceptions for the python-kraken-sdk

exception kraken.exceptions.**KrakenApiLimitExceededError**(**args*, ***kwargs*)

Bases: Exception

API rate limit exceeded. Please check your rate limits.

add_note()

Exception.add_note(note) – add a note to the exception

with_traceback()

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

exception kraken.exceptions.**KrakenAuthenticationError**(**args*, ***kwargs*)

Bases: Exception

Credentials are invalid.

add_note()

Exception.add_note(note) – add a note to the exception

with_traceback()

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

exception kraken.exceptions.**KrakenAuthenticationFailedError**(**args*, ***kwargs*)

Bases: Exception

The account or its permissions could not be authenticated.

add_note()

Exception.add_note(note) – add a note to the exception

with_traceback()

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

exception kraken.exceptions.**KrakenBadRequestError**(**args*, ***kwargs*)

Bases: Exception

The request payload is malformed, incorrect or ambiguous.

add_note()

Exception.add_note(note) – add a note to the exception

with_traceback()

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

exception kraken.exceptions.**KrakenCannotOpenPositionError**(**args*, ***kwargs*)

Bases: Exception

User/tier is ineligible for margin trading.

add_note()

Exception.add_note(note) – add a note to the exception

with_traceback()

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

exception kraken.exceptions.**KrakenCostMinimumNotMetError**(**args*, ***kwargs*)

Bases: Exception

Cost (price * volume) does not meet costmin.

add_note()

Exception.add_note(note) – add a note to the exception

with_traceback()

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

exception kraken.exceptions.**KrakenDeadlineElapsedError**(**args*, ***kwargs*)

Bases: Exception

The request timed out according to the default or specified deadline.

add_note()

Exception.add_note(note) – add a note to the exception

with_traceback()

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

exception kraken.exceptions.**KrakenInsufficientAvailableFundsError**(**args*, ***kwargs*)

Bases: Exception

Client does not have the necessary funds.

add_note()

Exception.add_note(note) – add a note to the exception

with_traceback()

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

exception kraken.exceptions.**KrakenInsufficientFundsError**(**args*, ***kwargs*)

Bases: Exception

Client does not have the necessary funds.

add_note()

Exception.add_note(note) – add a note to the exception

with_traceback()

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

exception kraken.exceptions.**KrakenInsufficientMarginError**(**args*, ***kwargs*)

Bases: Exception

Exchange does not have available funds for this margin trade.

add_note()

Exception.add_note(note) – add a note to the exception

with_traceback()

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

exception kraken.exceptions.KrakenInvalidAPIKeyError(*args, **kwargs)

Bases: Exception

An invalid API-Key header was supplied.

add_note()

Exception.add_note(note) – add a note to the exception

with_traceback()

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

exception kraken.exceptions.KrakenInvalidAccountError(*args, **kwargs)

Bases: Exception

The account is invalid.

add_note()

Exception.add_note(note) – add a note to the exception

with_traceback()

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

exception kraken.exceptions.KrakenInvalidAmountError(*args, **kwargs)

Bases: Exception

The specified amount is invalid.

add_note()

Exception.add_note(note) – add a note to the exception

with_traceback()

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

exception kraken.exceptions.KrakenInvalidArgumentsError(*args, **kwargs)

Bases: Exception

The request payload is malformed, incorrect or ambiguous.

add_note()

Exception.add_note(note) – add a note to the exception

with_traceback()

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

exception kraken.exceptions.KrakenInvalidArgumentsIndexUnavailableError(*args, **kwargs)

Bases: Exception

Index pricing is unavailable for stop/profit orders on this pair.

add_note()

Exception.add_note(note) – add a note to the exception

with_traceback()

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

exception kraken.exceptions.**KrakenInvalidNonceError**(**args*, ***kwargs*)

Bases: Exception

An invalid nonce was supplied.

add_note()

Exception.add_note(note) – add a note to the exception

with_traceback()

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

exception kraken.exceptions.**KrakenInvalidOrderError**(**args*, ***kwargs*)

Bases: Exception

Order is invalid.

add_note()

Exception.add_note(note) – add a note to the exception

with_traceback()

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

exception kraken.exceptions.**KrakenInvalidPriceError**(**args*, ***kwargs*)

Bases: Exception

Price is invalid.

add_note()

Exception.add_note(note) – add a note to the exception

with_traceback()

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

exception kraken.exceptions.**KrakenInvalidReferenceIdError**(**args*, ***kwargs*)

Bases: Exception

The requested reference id is invalid.

add_note()

Exception.add_note(note) – add a note to the exception

with_traceback()

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

exception kraken.exceptions.**KrakenInvalidSignatureError**(**args*, ***kwargs*)

Bases: Exception

An invalid API-Sign header was supplied.

add_note()

Exception.add_note(note) – add a note to the exception

with_traceback()

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

exception kraken.exceptions.**KrakenInvalidStakingMethodError**(**args*, ***kwargs*)

Bases: Exception

The staking method is invalid.

add_note()

Exception.add_note(note) – add a note to the exception

with_traceback()

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

exception kraken.exceptions.**KrakenInvalidUnitError**(**args*, ***kwargs*)

Bases: Exception

The specified unit is invalid.

add_note()

Exception.add_note(note) – add a note to the exception

with_traceback()

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

exception kraken.exceptions.**KrakenMarginAllowedExceededError**(**args*, ***kwargs*)

Bases: Exception

User has exceeded their margin allowance.

add_note()

Exception.add_note(note) – add a note to the exception

with_traceback()

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

exception kraken.exceptions.**KrakenMarginLevelToLowError**(**args*, ***kwargs*)

Bases: Exception

Client has insufficient equity or collateral.

add_note()

Exception.add_note(note) – add a note to the exception

with_traceback()

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

exception kraken.exceptions.**KrakenMarginPositionSizeExceededError**(**args*, ***kwargs*)

Bases: Exception

Client would exceed the maximum position size for this pair.

add_note()

Exception.add_note(note) – add a note to the exception

with_traceback()

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

exception kraken.exceptions.**KrakenMarketInOnlyCancelModeError**(**args*, ***kwargs*)

Bases: Exception

Request can't be made at this time. Please check system status.

add_note()

Exception.add_note(note) – add a note to the exception

with_traceback()

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

exception kraken.exceptions.**KrakenMarketInOnlyPostModeError**(**args*, ***kwargs*)

Bases: Exception

Request can't be made at this time. Please check system status.

add_note()

Exception.add_note(note) – add a note to the exception

with_traceback()

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

exception kraken.exceptions.**KrakenMaxFeeExceededError**(**args*, ***kwargs*)

Bases: Exception

The fee was higher than the defined maximum.

add_note()

Exception.add_note(note) – add a note to the exception

with_traceback()

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

exception kraken.exceptions.**KrakenNotFoundError**(**args*, ***kwargs*)

Bases: Exception

The resource is not found.

add_note()

Exception.add_note(note) – add a note to the exception

with_traceback()

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

exception kraken.exceptions.**KrakenOrderForEditNotFoundError**(**args*, ***kwargs*)

Bases: Exception

The order for edit could not be found.

add_note()

Exception.add_note(note) – add a note to the exception

with_traceback()

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

exception kraken.exceptions.**KrakenOrderLimitsExceededError**(**args*, ***kwargs*)

Bases: Exception

Order limits exceeded. Please check your open orders limit.

add_note()

Exception.add_note(note) – add a note to the exception

with_traceback()

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

exception kraken.exceptions.**KrakenOrderMinimumNotMetError**(**args*, ***kwargs*)

Bases: Exception

Order size does not meet ordermin.

add_note()

Exception.add_note(note) – add a note to the exception

with_traceback()

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

exception kraken.exceptions.**KrakenPermissionDeniedError**(**args*, ***kwargs*)

Bases: Exception

API key doesn't have permission to make this request.

add_note()

Exception.add_note(note) – add a note to the exception

with_traceback()

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

exception kraken.exceptions.**KrakenPositionLimitExceededError**(**args*, ***kwargs*)

Bases: Exception

Position limit exceeded. Please check your limits.

add_note()

Exception.add_note(note) – add a note to the exception

with_traceback()

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

exception kraken.exceptions.**KrakenRateLimitExceededError**(**args*, ***kwargs*)

Bases: Exception

API rate limit exceeded. Please check your rate limits.

add_note()

Exception.add_note(note) – add a note to the exception

with_traceback()

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

exception kraken.exceptions.**KrakenRequiredArgumentMissingError**(**args*, ***kwargs*)

Bases: Exception

The request is missing a required argument.

add_note()

Exception.add_note(note) – add a note to the exception

with_traceback()

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

exception kraken.exceptions.**KrakenServiceUnavailableError**(**args*, ***kwargs*)

Bases: Exception

The matching engine or API is offline.

add_note()

Exception.add_note(note) – add a note to the exception

with_traceback()

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

exception kraken.exceptions.**KrakenTemporaryLockoutError**(**args*, ***kwargs*)

Bases: Exception

The account was temporary locked out.

add_note()

Exception.add_note(note) – add a note to the exception

with_traceback()

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

exception kraken.exceptions.**KrakenTickSizeInvalidCheckError**(**args*, ***kwargs*)

Bases: Exception

Price submitted is not a valid multiple of the pair's tick_size.

add_note()

Exception.add_note(note) – add a note to the exception

with_traceback()

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

exception kraken.exceptions.**KrakenToManyAddressesError**(**args*, ***kwargs*)

Bases: Exception

To many addresses specified.

add_note()

Exception.add_note(note) – add a note to the exception

with_traceback()

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

exception kraken.exceptions.**KrakenUnavailableError**(**args*, ***kwargs*)

Bases: Exception

The requested resource is unavailable.

add_note()

Exception.add_note(note) – add a note to the exception

with_traceback()

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

exception kraken.exceptions.**KrakenUnknownAssetError**(**args*, ***kwargs*)

Bases: Exception

The asset is unknown.

add_note()

Exception.add_note(note) – add a note to the exception

with_traceback()

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

exception kraken.exceptions.**KrakenUnknownAssetPairError**(**args*, ***kwargs*)

Bases: Exception

The asset pair is unknown.

add_note()

Exception.add_note(note) – add a note to the exception

with_traceback()

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

exception kraken.exceptions.**KrakenUnknownMethodError**(**args*, ***kwargs*)

Bases: Exception

The endpoint or method is not known.

add_note()

Exception.add_note(note) – add a note to the exception

with_traceback()

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

exception kraken.exceptions.**KrakenUnknownOrderError**(**args*, ***kwargs*)

Bases: Exception

Order is unknown.

add_note()

Exception.add_note(note) – add a note to the exception

with_traceback()

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

exception kraken.exceptions.**KrakenUnknownPositionError**(**args*, ***kwargs*)

Bases: Exception

Position is unknown.

add_note()

Exception.add_note(note) – add a note to the exception

with_traceback()

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

exception kraken.exceptions.**KrakenUnknownReferenceIdError**(**args*, ***kwargs*)

Bases: Exception

The requested reference id is unknown.

add_note()

Exception.add_note(note) – add a note to the exception

with_traceback()

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

exception kraken.exceptions.**KrakenUnknownWithdrawKeyError**(**args*, ***kwargs*)

Bases: Exception

The requested withdrawal key is unknown.

add_note()

Exception.add_note(note) – add a note to the exception

with_traceback()

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

exception kraken.exceptions.**MaxReconnectError**(**args*, ***kwargs*)

Bases: Exception

To many reconnect tries.

add_note()

Exception.add_note(note) – add a note to the exception

with_traceback()

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

CHAPTER

ELEVEN

KNOWN ISSUES

Issues listed here: python-kraken-sdk/issues

11.1 Futures Trading

- The Kraken API returns 500 - INTERNAL_SERVER_ERROR for some endpoints if `order_id` or `orderId`, `cliOrdId` seems to work in all cases.
- Kraken's API doesn't seem to know the `trailing_stop` order type and raises an error if this type is part of an order. This order type is documented here https://docs.kraken.com/api/docs/futures-api/trading/send-order

Listing 1: `trailing_stop` order type not working in Kraken Futures

```python
from kraken.futures import Trade

Trade(key="api-key", secret="secret-key").create_order(
    orderType="trailing_stop",
    size=10,
    side="buy",
    symbol="PI_XBTUSD",
    limitPrice=12000,
    triggerSignal="mark",
    trailingStopDeviationUnit="PERCENT",
    trailingStopMaxDeviation=10,
)
""" Output:
{
    "status": "BAD_REQUEST",
    "result": "error",
    "errors": [{
        "code": 11,
        "message": "invalid order type"
    }],
    "serverTime":"2023-04-07T19:26:41.299Z"
}
"""
```

CHAPTER

TWELVE

LICENSE

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

   "License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

   "Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

   "Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

   "You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

   "Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

   "Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

   "Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

   "Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

   "Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its

representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

   (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

   You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

> To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "{}" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

CHAPTER

THIRTEEN

INDICES AND TABLES

- genindex

PYTHON MODULE INDEX

INDEX

Symbols

A

C

D

E

F

G

I

K

L

M

O

P

R

S

T

U

W