BinanceWatch

Release 0.1.4

EtWnn

Apr 27, 2021

CONTENTS

CHAPTER

ONE

NOTE

This library is under development by EtWnn, feel free to drop your suggestions or remarks in the discussion tab of the git repo. You are also welcome to contribute by submitting PRs.

This is an unofficial tracker for binance accounts. I am in no way affiliated with Binance, use at your own risk.

Source Code: https://github.com/EtWnn/BinanceWatch

Documentation: https://binancewatch.readthedocs.io

CHAPTER

TWO

FEATURES

If you used quite intensively Binance, it can take some time to retrieve everything that happened on your account. This library is made to save locally the events of your account so that you don't need to fetch your history from the beginning every time.

It currently supports:

- Spot Trades
- Spot Crypto Deposits
- Spot Crypto Withdraws
- Spot Dividends
- Spot Dusts
- Universal Transfers
- Lending Purchases
- Lending Interests
- Lending Redemptions
- Cross Margin Trades
- Cross Margin Repayment
- Cross Margin Loans
- Cross Margin Interests
- Isolated Margin Trades
- Isolated Margin Repayment
- Isolated Margin Loans
- Isolated Margin Interests
- Isolated Margin Transfers *

*: see `Known Issues` section below.

CHAPTER

THREE

QUICK TOUR

Generate an API Key in your binance account. Only read permissions are needed.

`BinanceWatch` is available on PYPI, install with `pip`:

```
pip install BinanceWatch
```

If you prefer to install the latest developments use:

```
pip install git+https://github.com/EtWnn/BinanceWatch.git@develop
```

Use your Binance api keys to initiate the manager:

```
from BinanceWatch.BinanceManager import BinanceManager

api_key = "<API_KEY>"
api_secret = "<API_SECRET>"

bm = BinanceManager(api_key, api_secret)

# fetch the latest spot trades from Binance
bm.update_all_spot_trades()
```

```
Out -> fetching BIFIBUSD: 100%|| 1349/1349 [06:24<00:00,  3.51it/s]
```

```
from datetime import datetime
from BinanceWatch.utils.time_utils import datetime_to_millistamp


start_time = datetime_to_millistamp(datetime(2018,1,1))

# get the locally saved spot trades made after 2018/01/01
spot_trades = bm.db.get_trades('spot', start_time=start_time)
```

You can also call update functions at an account-type level, and it will call every update methods related to this account-type:

```
bm.update_spot()  # (trades, transfers, deposits ...)

bm.update_cross_margin()  # (trades, loans, repays, interests...)

bm.update_lending()  # (purchases, interests, redemptions..)
```

CHAPTER

FOUR

DONATION

If this library has helped you in any way, feel free to donate:

- **BTC**: 14ou4fMYoMVYbWEKnhADPJUNVytWQWx9HG
- **ETH**: 0xfb0ebcf8224ce561bfb06a56c3b9a43e1a4d1be2
- **LTC**: LfHgc969RFUjnmyLn41SRDvmT146jUg9tE
- **EGLD**: erd1qk98xm2hgztvmq6s4jwtk06g6laattewp6vh20z393drzy5zzfrq0gaefh

CHAPTER

FIVE

KNOWN ISSUES:

Some endpoints are not yet provided by Binance, so they can't be implemented in this library:

- Fiat withdraws and deposits
- Locked stacking history
- Direct purchases with debit card
- Some isolated margin transfers are not picked up by the API, the reason is unknown at the moment (**I am looking for testers**)

5.1 Contents

5.1.1 Getting Started

Installation

`BinanceWatch` is available on PYPI, install with `pip`:

```
pip install BinanceWatch
```

Register on Binance

If you are interested in this library, I assume that you have already a Binance account. If not you can register an account with Binance.

Generate an API Key

To use signed account methods you are required to create an API Key. In this library, only read permissions are needed so don't forget to disabled the others restrictions (trading, withdrawal . . .)

Initialise the manager

Pass your API Key and Secret to the manager

```
from BinanceWatch.BinanceManager import BinanceManager
bm = BinanceManager(api_key, api_secret)
```

API calls

All the API calls to Binance are handled by the library python-binance. Don't hesitate to check their very useful githib repo and drop a star!

Updates

The manager is mainly used to update the transactions saved locally. By calling an update method from the manager, it will check if any transaction has been made between the last one saved locally and the current time. The details of the update methods are in the section *Binance Manager*.

Retrievals

Each manager has a database, which is where the results of the Binance API calls are stored. By calling the get methods of the database, you will retrieve the history of your Binance account. See the *Binance DataBase* section for more details.

Examples

You can updates the elements by type, for example here with the crypto spot deposits:

```
bm.update_spot_deposits()  # will fetch the latest deposits not saved locally

bm.db.get_spot_deposits()  # return the deposits saved locally
```

```
[
    ('azdf5e6a1d5z',     # transaction id
    1589479004000,       # deposit time
    'LTC',               # asset
    14.25),              # amount
...
]
```

You can also use larger update methods, that will update several types of elements. Below the method will update every elements of a cross margin account:

```
bm.update_cross_margin()  # will fetch the latest transfers, trades, loans ...

bm.db.get_trades(trade_type='cross_margin')
```

```
[
    (384518832,          # trade_id
    1582892988052,       # trade time
    'BTC',               # asset
```

(continues on next page)

(continued from previous page)

```
    'USDT',             # ref asset
    0.0015,             # asset quantity
    9011.2,             # asset price to ref asset
    0.01425,            # fee
    'USDT',             # fee asset
    0),                 # is_buyer
...
]
```

5.1.2 Binance Manager

class BinanceWatch.BinanceManager.**BinanceManager**(*api_key: str*, *api_secret: str*, *account_name: str = 'default'*)

This class is in charge of filling the database by calling the binance API

API_MAX_RETRY = 3

__init__(*api_key: str*, *api_secret: str*, *account_name: str = 'default'*)

Initialise the binance manager.

Parameters

- **api_key** (*str*) – key for the Binance api
- **api_secret** (*str*) – secret for the Binance api
- **account_name** (*str*) – if you have several accounts to monitor, you need to give them different names or the database will collide

get_margin_symbol_info(*isolated: bool*) → List[Dict]

Return information about margin symbols as provided by the binance API

sources: https://binance-docs.github.io/apidocs/spot/en/#get-all-isolated-margin-symbol-user_data https://binance-docs.github.io/apidocs/spot/en/#get-all-cross-margin-pairs-market_data

Parameters **isolated** (*bool*) – If isolated data are to be returned, otherwise it will be cross margin data

Returns Info on the trading symbols

Return type List[Dict]

```
# cross margin
[
    {
        'id': 351637150141315861,
        'symbol': 'BNBBTC',
        'base': 'BNB',
        'quote': 'BTC',
        'isMarginTrade': True,
        'isBuyAllowed': True,
        'isSellAllowed': True
    },
    ...
]

# isolated margin
[
```

(continues on next page)

(continued from previous page)

```
    {
        'symbol': '1INCHBTC',
        'base': '1INCH',
        'quote': 'BTC',
        'isMarginTrade': True,
        'isBuyAllowed': True,
        'isSellAllowed': True
    },
    ...
]
```

update_all_cross_margin_trades(*limit: int = 1000*)
This update the cross margin trades in the database for every trading pairs

Parameters **limit** (*int*) – max size of each trade requests

Returns None

Return type None

update_all_spot_trades(*limit: int = 1000*)
This update the spot trades in the database for every trading pairs

Parameters **limit** (*int*) – max size of each trade requests

Returns None

Return type None

update_cross_margin()
call all update methods related to cross margin account

Returns None

Return type None

update_cross_margin_loans()
update the loans for all cross margin assets

Returns None

Return type None

update_cross_margin_repays()
update the repays for all cross margin assets

Returns None

Return type None

update_isolated_margin()
call all update methods related to isolated margin account

Returns None

Return type None

update_isolated_margin_interests(*symbols_info: Optional[List[Dict]] = None*)
Update the interests for isolated margin assets

Parameters **symbols_info** (*Optional[List[Dict]]*) – details on the symbols to fetch repays on. Each dictionary needs the fields 'asset' and 'ref_asset'. If not provided, will update all isolated symbols.

Returns None

Return type None

update_isolated_margin_loans(*symbols_info: Optional[List[Dict]] = None*)
Update the loans for isolated margin assets

Parameters **symbols_info** (*Optional[List[Dict]]*) – details on the symbols to fetch loans on. Each dictionary needs the fields 'asset' and 'ref_asset'. If not provided, will update all isolated symbols.

Returns None

Return type None

update_isolated_margin_repays(*symbols_info: Optional[List[Dict]] = None*)
Update the repays for isolated margin assets

Parameters **symbols_info** (*Optional[List[Dict]]*) – details on the symbols to fetch repays on. Each dictionary needs the fields 'asset' and 'ref_asset'. If not provided, will update all isolated symbols.

Returns None

Return type None

update_isolated_margin_trades(*symbols_info: Optional[List[Dict]] = None*)
This update the isolated margin trades in the database for every trading pairs

Parameters **symbols_info** (*Optional[List[Dict]]*) – details on the symbols to fetch trades on. Each dictionary needs the fields 'asset' and 'ref_asset'. If not provided, will update all isolated symbols.

Returns None

Return type None

update_isolated_margin_transfers(*symbols_info: Optional[List[Dict]] = None*)
Update the transfers to and from isolated symbols

Parameters **symbols_info** (*Optional[List[Dict]]*) – details on the symbols to fetch repays on. Each dictionary needs the fields 'asset' and 'ref_asset'. If not provided, will update all isolated symbols.

Returns None

Return type None

update_isolated_symbol_transfers(*isolated_symbol: str*)
Update the transfers made to and from an isolated margin symbol

sources: https://binance-docs.github.io/apidocs/spot/en/#get-isolated-margin-transfer-history-user_data

Parameters **isolated_symbol** (*str*) – isolated margin symbol of trading

Returns

Return type

update_lending()
call all update methods related to lending activities

Returns None

Return type None

update_lending_interests()
update the lending interests database.

sources: https://python-binance.readthedocs.io/en/latest/binance.html#binance.client.Client.get_lending_interest_history https://binance-docs.github.io/apidocs/spot/en/#get-interest-history-user_data-2

Returns None

Return type None

update_lending_purchases()
update the lending purchases database.

sources: https://python-binance.readthedocs.io/en/latest/binance.html#binance.client.Client.get_lending_purchase_history https://binance-docs.github.io/apidocs/spot/en/#get-purchase-record-user_data

Returns None

Return type None

update_lending_redemptions()
update the lending redemptions database.

sources: https://python-binance.readthedocs.io/en/latest/binance.html#binance.client.Client.get_lending_redemption_history https://binance-docs.github.io/apidocs/spot/en/#get-redemption-record-user_data

Returns None

Return type None

update_margin_asset_loans(*asset: str*, *isolated_symbol: Optional[str] = None*)
update the loans database for a specified asset.

sources: https://binance-docs.github.io/apidocs/spot/en/#query-loan-record-user_data https://python-binance.readthedocs.io/en/latest/binance.html#binance.client.Client.get_margin_loan_details

Parameters

- **asset** (*str*) – asset for the loans
- **isolated_symbol** (*Optional[str]*) – only for isolated margin, provide the trading symbol. Otherwise cross margin data will be updated

Returns None

Return type None

update_margin_asset_repay(*asset: str*, *isolated_symbol: Optional[str] = None*)
update the repays database for a specified asset.

sources: https://binance-docs.github.io/apidocs/spot/en/#query-repay-record-user_data https://python-binance.readthedocs.io/en/latest/binance.html#binance.client.Client.get_margin_repay_details

Parameters

- **asset** (*str*) – asset for the repays
- **isolated_symbol** (*Optional[str]*) – only for isolated margin, provide the trading symbol. Otherwise cross margin data will be updated

Returns None

Return type None

update_margin_interests(*isolated_symbol: Optional[str] = None*, *show_pbar: bool = True*)
Update the interests for all cross margin assets or for a isolated margin symbol if provided.

sources: https://binance-docs.github.io/apidocs/spot/en/#query-repay-record-user_data

Parameters

- **isolated_symbol** (*Optional[str]*) – only for isolated margin, provide the trading symbol. Otherwise cross margin data will be updated
- **show_pbar** (*bool*) – if the progress bar is displayed

Returns

Return type

update_margin_symbol_trades(*asset: str*, *ref_asset: str*, *is_isolated: bool = False*, *limit: int = 1000*)

This update the margin trades in the database for a single trading pair. It will check the last trade id and will requests the all trades after this trade_id.

sources: https://binance-docs.github.io/apidocs/spot/en/#query-margin-account-39-s-trade-list-user_data https://python-binance.readthedocs.io/en/latest/binance.html#binance.client.Client.get_margin_trades

Parameters

- **asset** (*string*) – name of the asset in the trading pair (ex 'BTC' for 'BTCUSDT')
- **ref_asset** (*string*) – name of the reference asset in the trading pair (ex 'USDT' for 'BTCUSDT')
- **is_isolated** (*bool*) – if margin type is isolated, default False
- **limit** (*int*) – max size of each trade requests

Returns None

Return type None

update_spot()

call all update methods related to the spot account

Returns None

Return type None

update_spot_deposits(*day_jump: float = 90*)

This fetch the crypto deposit made on the spot account from the last deposit time in the database to now. It is done with multiple call, each having a time window of day_jump days. The deposits are then saved in the database. Only successful deposits are fetched.

sources: https://python-binance.readthedocs.io/en/latest/binance.html#binance.client.Client.get_deposit_history https://binance-docs.github.io/apidocs/spot/en/#deposit-history-user_data

Parameters **day_jump** (*float*) – length of the time window for each call (max 90)

Returns None

Return type None

update_spot_dividends(*day_jump: float = 90*, *limit: int = 500*)

update the dividends database (earnings distributed by Binance) sources: https://python-binance.readthedocs.io/en/latest/binance.html#binance.client.Client.get_asset_dividend_history https://binance-docs.github.io/apidocs/spot/en/#asset-dividend-record-user_data

Parameters

- **day_jump** (*float*) – length of the time window in days, max is 90
- **limit** (*int*) – max number of dividends to retrieve per call, max is 500

Returns None

Return type None

update_spot_dusts()
update the dust database. As there is no way to get the dust by id or timeframe, the table is cleared for each update

sources: https://python-binance.readthedocs.io/en/latest/binance.html#binance.client.Client.get_dust_log https://binance-docs.github.io/apidocs/spot/en/#dustlog-user_data

Returns None

Return type None

update_spot_symbol_trades(*asset: str*, *ref_asset: str*, *limit: int = 1000*)
This update the spot trades in the database for a single trading pair. It will check the last trade id and will requests the all trades after this trade_id.

sources: https://python-binance.readthedocs.io/en/latest/binance.html#binance.client.Client.get_my_trades https://binance-docs.github.io/apidocs/spot/en/#account-trade-list-user_data

Parameters

- **asset** (*string*) – name of the asset in the trading pair (ex 'BTC' for 'BTCUSDT')
- **ref_asset** (*string*) – name of the reference asset in the trading pair (ex 'USDT' for 'BTCUSDT')
- **limit** (*int*) – max size of each trade requests

Returns None

Return type None

update_spot_withdraws(*day_jump: float = 90*)
This fetch the crypto withdraws made on the spot account from the last withdraw time in the database to now. It is done with multiple call, each having a time window of day_jump days. The withdraws are then saved in the database. Only successful withdraws are fetched.

sources: https://python-binance.readthedocs.io/en/latest/binance.html#binance.client.Client.get_withdraw_history https://binance-docs.github.io/apidocs/spot/en/#withdraw-history-user_data

Parameters **day_jump** (*float*) – length of the time window for each call (max 90)

Returns None

Return type None

update_universal_transfers(*transfer_filter: Optional[str] = None*)
update the universal transfers database.

sources: https://python-binance.readthedocs.io/en/latest/binance.html#binance.client.Client.query_universal_transfer_history https://binance-docs.github.io/apidocs/spot/en/#query-user-universal-transfer-history

Parameters **transfer_filter** (*Optional[str]*) – if not None, only the transfers containing this filter will be updated (ex: 'MAIN')

Returns None

Return type None

5.1.3 Binance DataBase

class BinanceWatch.storage.BinanceDataBase.**BinanceDataBase**(*name: str = 'binance_db'*)

Bases: *BinanceWatch.storage.DataBase.DataBase*

Handles the recording of the binance account in a local database

__init__(*name: str = 'binance_db'*)

Initialise a binance database instance

Parameters **name** (*str*) – name of the database

add_deposit(*tx_id: str*, *insert_time: int*, *amount: float*, *asset: str*, *auto_commit=True*)

Add a deposit to the database

Parameters

- **tx_id** (*str*) – transaction id
- **insert_time** (*int*) – millistamp when the deposit arrived on binance
- **amount** (*float*) – amount of token deposited
- **asset** (*str*) – name of the token
- **auto_commit** (*bool*) – if the database should commit the change made, default True

Returns None

Return type None

add_dividend(*div_id: int*, *div_time: int*, *asset: str*, *amount: float*, *auto_commit: bool = True*)

Add a dividend to the database

Parameters

- **div_id** (*int*) – dividend id
- **div_time** (*int*) – millistamp of dividend reception
- **asset** (*str*) – name of the dividend unit
- **amount** (*float*) – amount of asset distributed
- **auto_commit** (*bool*) – if the database should commit the change made, default True

Returns None

Return type None

add_isolated_transfer(*transfer_id: int*, *transfer_type: str*, *transfer_time: int*, *isolated_symbol: str*, *asset: str*, *amount: float*, *auto_commit: bool = True*)

Add a universal transfer to the database

Parameters

- **transfer_id** (*int*) – id of the transfer
- **transfer_type** (*str*) – enum of the transfer type (ex: 'MAIN_MARGIN')
- **transfer_time** (*int*) – millistamp of the operation
- **isolated_symbol** (*str*) – isolated symbol that received or sent the transfer
- **asset** (*str*) – asset that got transferred
- **amount** (*float*) – amount transferred

- **auto_commit** (*bool*) – if the database should commit the change made, default True

Returns None

Return type None

add_lending_interest(*time: int*, *lending_type: str*, *asset: str*, *amount: float*, *auto_commit: bool = True*)

Add an lending interest to the database

Parameters

- **time** (*int*) – millitstamp of the operation
- **lending_type** (*str*) – either 'DAILY', 'ACTIVITY' or 'CUSTOMIZED_FIXED'
- **asset** (*str*) – asset that was received
- **amount** (*float*) – amount of asset received
- **auto_commit** (*bool*) – if the database should commit the change made, default True

Returns None

Return type None

add_lending_purchase(*purchase_id: int*, *purchase_time: int*, *lending_type: str*, *asset: str*, *amount: float*, *auto_commit: bool = True*)

Add a lending purchase to the database

Parameters

- **purchase_id** (*int*) – id of the purchase
- **purchase_time** (*int*) – millitstamp of the operation
- **lending_type** (*str*) – either 'DAILY', 'ACTIVITY' or 'CUSTOMIZED_FIXED'
- **asset** (*str*) – asset lent
- **amount** (*float*) – amount of asset lent
- **auto_commit** (*bool*) – if the database should commit the change made, default True

Returns None

Return type None

add_lending_redemption(*redemption_time: int*, *lending_type: str*, *asset: str*, *amount: float*, *auto_commit: bool = True*)

Add a lending redemption to the database

Parameters

- **redemption_time** (*int*) – millitstamp of the operation
- **lending_type** (*str*) – either 'DAILY', 'ACTIVITY' or 'CUSTOMIZED_FIXED'
- **asset** (*str*) – asset lent
- **amount** (*float*) – amount of asset redeemed
- **auto_commit** (*bool*) – if the database should commit the change made, default True

Returns None

Return type None

add_loan(*tx_id: int*, *loan_time: int*, *asset: str*, *principal: float*, *isolated_symbol: Optional[str] = None*, *auto_commit: bool = True*)
Add a loan to the database

Parameters

- **tx_id** (*int*) – binance id for the transaction (uniqueness?)
- **loan_time** (*int*) – millitstamp of the operation
- **asset** (*str*) – asset that got loaned
- **principal** (*float*) – amount of loaned asset
- **isolated_symbol** (*Optional[str]*) – for isolated margin, provided the trading symbol otherwise it will be counted a cross margin data
- **auto_commit** (*bool*) – if the database should commit the change made, default True

Returns None

Return type None

add_margin_interest(*interest_time: int*, *asset: str*, *interest: float*, *interest_type: str*, *isolated_symbol: Optional[str] = None*, *auto_commit: bool = True*)
Add a margin interest to the database

Parameters

- **interest_time** (*int*) – millistamp of the operation
- **asset** (*str*) – asset that got repaid
- **interest** (*float*) – amount of interest accrued
- **interest_type** (*str*) – one of (PERIODIC, ON_BORROW, PERIODIC_CONVERTED, ON_BORROW_CONVERTED)
- **isolated_symbol** (*Optional[str]*) – for isolated margin, provided the trading symbol otherwise it will be counted a cross margin data
- **auto_commit** (*bool*) – if the database should commit the change made, default True

Returns None

Return type None

add_repay(*tx_id: int*, *repay_time: int*, *asset: str*, *principal: float*, *interest: float*, *isolated_symbol: Optional[str] = None*, *auto_commit: bool = True*)
Add a repay to the database

Parameters

- **tx_id** (*int*) – binance id for the transaction (uniqueness?)
- **repay_time** (*int*) – millitstamp of the operation
- **asset** (*str*) – asset that got repaid
- **principal** (*float*) – principal amount repaid for the loan
- **interest** (*float*) – amount of interest repaid for the loan
- **isolated_symbol** (*Optional[str]*) – for isolated margin, provided the trading symbol otherwise it will be counted a cross margin data
- **auto_commit** (*bool*) – if the database should commit the change made, default True

Returns None

Return type None

add_spot_dust(*tran_id: str*, *time: int*, *asset: str*, *asset_amount: float*, *bnb_amount: float*, *bnb_fee: float*, *auto_commit: bool = True*)

Add dust operation to the database

Parameters

- **tran_id** (*str*) – id of the transaction (non unique)
- **time** (*int*) – millitstamp of the operation
- **asset** (*str*) – asset that got converted to BNB
- **asset_amount** (*float*) – amount of asset that got converted
- **bnb_amount** (*float*) – amount received from the conversion
- **bnb_fee** (*float*) – fee amount in BNB
- **auto_commit** (*bool*) – if the database should commit the change made, default True

Returns None

Return type None

add_trade(*trade_type: str*, *trade_id: int*, *trade_time: int*, *asset: str*, *ref_asset: str*, *qty: float*, *price: float*, *fee: float*, *fee_asset: str*, *is_buyer: bool*, *symbol: Optional[str] = None*, *auto_commit: bool = True*)

Add a trade to the database

Parameters

- **trade_type** (*string, must be one of {'spot', 'cross_margin', 'isolated_margin'}*) – type trade executed
- **trade_id** (*int*) – id of the trade (binance id, unique per trading pair)
- **trade_time** (*int*) – millistamp of the trade
- **asset** (*string*) – name of the asset in the trading pair (ex 'BTC' for 'BTCUSDT')
- **ref_asset** (*string*) – name of the reference asset in the trading pair (ex 'USDT' for 'BTCUSDT')
- **qty** (*float*) – quantity of asset exchanged
- **price** (*float*) – price of the asset regarding the ref_asset
- **fee** (*float*) – amount kept by the exchange
- **fee_asset** (*str*) – token unit for the fee
- **is_buyer** (*bool*) – if the trade is a buy or a sell
- **symbol** (*Optional[str]*) – trading symbol, mandatory if thr trade_type is isolated margin
- **auto_commit** (*bool*) – if the database should commit the change made, default True

Returns None

Return type None

add_universal_transfer(*transfer_id: int*, *transfer_type: str*, *transfer_time: int*, *asset: str*, *amount: float*, *auto_commit: bool = True*)

Add a universal transfer to the database

Parameters

- **transfer_id** (*int*) – id of the transfer
- **transfer_type** (*str*) – enum of the transfer type (ex: 'MAIN_MARGIN')
- **transfer_time** (*int*) – millistamp of the operation
- **asset** (*str*) – asset that got transferred
- **amount** (*float*) – amount transferred
- **auto_commit** (*bool*) – if the database should commit the change made, default True

Returns None

Return type None

add_withdraw(*withdraw_id: str*, *tx_id: str*, *apply_time: int*, *asset: str*, *amount: float*, *fee: float*, *auto_commit: bool = True*)

Add a withdraw to the database

Parameters

- **withdraw_id** (*str*) – binance if of the withdraw
- **tx_id** (*str*) – transaction id
- **apply_time** (*int*) – millistamp when the withdraw was requested
- **asset** (*str*) – name of the token
- **amount** (*float*) – amount of token withdrawn
- **fee** (*float*) – amount of the asset paid for the withdraw
- **auto_commit** (*bool*) – if the database should commit the change made, default True

Returns None

Return type None

get_isolated_transfers(*isolated_symbol: Optional[str] = None*, *start_time: Optional[int] = None*, *end_time: Optional[int] = None*)

Return isolated transfers stored in the database. isolated_symbol and time filters can be used

Parameters

- **isolated_symbol** (*Optional[str]*) – for isolated margin, provided the trading symbol otherwise it will be counted a cross margin data
- **start_time** (*Optional[int]*) – fetch only transfers after this millistamp
- **end_time** (*Optional[int]*) – fetch only transfers before this millistamp

Returns The raw rows selected as saved in the database

Return type List[Tuple]

```
[
    (1206491332,        # transfer id
    'IN',               # transfer type (IN or OUT)
    1589121841000,      # time
    'BTCBUSD',          # isolated symbol
    'BTC',              # asset
    10.594112),         # amount
]
```

get_last_isolated_transfer_time(*isolated_symbol: str*) → int
Return the latest time when a isolated margin transfer was made If None, return the millistamp corresponding to 2017/01/01

Parameters **isolated_symbol** (*str*) – isolated symbol that received or sent the transfers

Returns millistamp

Return type int

get_last_lending_interest_time(*lending_type: Optional[str] = None*) → int
Return the latest time when an interest was received. If None, return the millistamp corresponding to 2017/01/01

Parameters **lending_type** (*str*) – type of lending

Returns millistamp

Return type int

get_last_lending_purchase_time(*lending_type: Optional[str] = None*) → int
Return the latest time when an lending purchase was made. If None, return the millistamp corresponding to 2017/01/01

Parameters **lending_type** (*str*) – type of lending

Returns millistamp

Return type int

get_last_lending_redemption_time(*lending_type: Optional[str] = None*) → int
Return the latest time when an lending redemption was made. If None, return the millistamp corresponding to 2017/01/01

Parameters **lending_type** (*str*) – type of lending

Returns millistamp

Return type int

get_last_loan_time(*asset: str*, *isolated_symbol: Optional[str] = None*) → int
Return the latest time when an loan was made on a defined asset If None, return the millistamp corresponding to 2017/01/01

Parameters

- **asset** (*str*) – name of the asset loaned
- **isolated_symbol** (*Optional[str]*) – only for isolated margin, provide the trading symbol (otherwise cross data are returned)

Returns millistamp

Return type int

get_last_margin_interest_time(*asset: Optional[str] = None*, *isolated_symbol: Optional[str] = None*) → int
Return the latest time when a margin interest was accured on a defined asset or on all assets If None, return the millistamp corresponding to 2017/01/01

Parameters

- **asset** (*Optional[str]*) – name of the asset charged as interest
- **isolated_symbol** (*Optional[str]*) – only for isolated margin, provide the trading symbol (otherwise cross data are returned)

Returns millistamp

Return type int

get_last_repay_time(*asset: str*, *isolated_symbol: Optional[str] = None*) → int
Return the latest time when a repay was made on a defined asset If None, return the millistamp corresponding to 2017/01/01

Parameters

- **asset** (*str*) – name of the asset repaid
- **isolated_symbol** (*Optional[str]*) – only for isolated margin, provide the trading symbol (otherwise cross data are returned)

Returns millistamp

Return type int

get_last_spot_deposit_time() → int
Fetch the latest time a deposit has been made on the spot account. If None is found, return the millistamp corresponding to 2017/1/1

Returns last deposit millistamp

Return type int

get_last_spot_dividend_time() → int
Fetch the latest time a dividend has been distributed on the spot account. If None is found, return the millistamp corresponding to 2017/1/1

Returns

get_last_spot_withdraw_time() → int
Fetch the latest time a withdraw has been made on the spot account. If None is found, return the millistamp corresponding to 2017/1/1

Returns

get_last_universal_transfer_time(*transfer_type: str*) → int
Return the latest time when a universal transfer was made If None, return the millistamp corresponding to 2017/01/01

Parameters **transfer_type** (*str*) – enum of the transfer type (ex: 'MAIN_MARGIN')

Returns millistamp

Return type int

get_lending_interests(*lending_type: Optional[str] = None*, *asset: Optional[str] = None*, *start_time: Optional[int] = None*, *end_time: Optional[int] = None*)
Return lending interests stored in the database. Asset type and time filters can be used

Parameters

- **lending_type** (*Optional[str]*) – fetch only interests from this lending type
- **asset** (*Optional[str]*) – fetch only interests from this asset
- **start_time** (*Optional[int]*) – fetch only interests after this millistamp
- **end_time** (*Optional[int]*) – fetch only interests before this millistamp

Returns The raw rows selected as saved in the database

Return type List[Tuple]

```
[
    (1619846515000,      # time
    'DAILY',             # lending type
    'DOT',               # asset
    0.00490156)          # amount
]
```

get_lending_purchases(*lending_type: Optional[str] = None*, *asset: Optional[str] = None*, *start_time: Optional[int] = None*, *end_time: Optional[int] = None*)

Return lending purchases stored in the database. Asset type and time filters can be used

Parameters

- **lending_type** (*Optional[str]*) – fetch only purchases from this lending type
- **asset** (*Optional[str]*) – fetch only purchases from this asset
- **start_time** (*Optional[int]*) – fetch only purchases after this millistamp
- **end_time** (*Optional[int]*) – fetch only purchases before this millistamp

Returns The raw rows selected as saved in the database

Return type List[Tuple]

```
[
    (58516828,           # purchase id
    1612841562000,       # time
    'DAILY',             # lending type
    'LTC',               # asset
    1.89151684),         # amount
]
```

get_lending_redemptions(*lending_type: Optional[str] = None*, *asset: Optional[str] = None*, *start_time: Optional[int] = None*, *end_time: Optional[int] = None*)

Return lending redemptions stored in the database. Asset type and time filters can be used

Parameters

- **lending_type** (*Optional[str]*) – fetch only redemptions from this lending type
- **asset** (*Optional[str]*) – fetch only redemptions from this asset
- **start_time** (*Optional[int]*) – fetch only redemptions after this millistamp
- **end_time** (*Optional[int]*) – fetch only redemptions before this millistamp

Returns The raw rows selected as saved in the database

Return type List[Tuple]

```
[
    1612841562000,       # time
    'DAILY',             # lending type
    'LTC',               # asset
    1.89151684),         # amount
]
```

get_loans(*margin_type: str*, *asset: Optional[str] = None*, *isolated_symbol: Optional[str] = None*, *start_time: Optional[int] = None*, *end_time: Optional[int] = None*)

Return loans stored in the database. Asset type and time filters can be used

Parameters

- **margin_type** – either 'cross' or 'isolated'
- **asset** (*Optional[str]*) – fetch only loans of this asset
- **isolated_symbol** (*Optional[str]*) – only for isolated margin, provide the trading symbol
- **start_time** (*Optional[int]*) – fetch only loans after this millistamp
- **end_time** (*Optional[int]*) – fetch only loans before this millistamp

Returns The raw rows selected as saved in the database

Return type List[Tuple]

```
# cross margin
[
    (8289451654,         # transaction id
    1559415215400,       # time
    'USDT',              # asset
    145.5491462),        # amount
]

# isolated margin
[
    (8289451654,         # transaction id
    1559415215400,       # time
    'BTCUSDT',           # symbol
    'USDT',              # asset
    145.5491462),        # amount
]
```

get_margin_interests(*margin_type: str*, *asset: Optional[str] = None*, *isolated_symbol: Optional[str] = None*, *start_time: Optional[int] = None*, *end_time: Optional[int] = None*)

Return margin interests stored in the database. Asset type and time filters can be used

Parameters

- **margin_type** – either 'cross' or 'isolated'
- **asset** (*Optional[str]*) – fetch only interests in this asset
- **isolated_symbol** (*Optional[str]*) – only for isolated margin, provide the trading symbol (otherwise cross data are returned)
- **start_time** (*Optional[int]*) – fetch only interests after this millistamp
- **end_time** (*Optional[int]*) – fetch only interests before this millistamp

Returns The raw rows selected as saved in the database

Return type List[Tuple]

```
# cross margin
[
    1559415215400,              # time
    'BNB',                      # asset
    0.51561,                    # interest
    'PERIODIC_CONVERTED'),      # interest type
]

# isolated margin
```

(continues on next page)

(continued from previous page)

```
[
    1559415215400,              # time
    'BTCBUSD',                  # symbol
    'BUSD',                     # asset
    0.51561,                    # interest
    'PERIODIC'),                # interest type
]
```

get_max_trade_id(*asset: str*, *ref_asset: str*, *trade_type: str*) → int

Return the latest trade id for a trading pair. If none is found, return -1

Parameters

- **asset** (*string*) – name of the asset in the trading pair (ex 'BTC' for 'BTCUSDT')
- **ref_asset** (*string*) – name of the reference asset in the trading pair (ex 'USDT' for 'BTCUSDT')
- **trade_type** (*string, must be one of {'spot', 'cross_margin'}*) – type trade executed

Returns latest trade id

Return type int

get_repays(*margin_type: str*, *asset: Optional[str] = None*, *isolated_symbol: Optional[str] = None*, *start_time: Optional[int] = None*, *end_time: Optional[int] = None*)

Return repays stored in the database. Asset type and time filters can be used

Parameters

- **margin_type** (*str*) – either 'cross' or 'isolated'
- **asset** (*Optional[str]*) – fetch only repays of this asset
- **isolated_symbol** (*Optional[str]*) – only for isolated margin, provide the trading symbol
- **start_time** (*Optional[int]*) – fetch only repays after this millistamp
- **end_time** (*Optional[int]*) – fetch only repays before this millistamp

Returns The raw rows selected as saved in the database

Return type List[Tuple]

```
# cross margin
[
    (8289451654,        # transaction id
    1559415215400,      # time
    'USDT',             # asset
    145.5491462,        # principal
    0.51561),           # interest
]

# isolated margin
[
    (8289451654,        # transaction id
    1559415215400,      # time
    'BTCUSDT',          # isolated symbol
    'USDT',             # asset
    145.5491462,        # principal
```

(continues on next page)

(continued from previous page)

```
    0.51561),           # interest
]
```

get_spot_deposits(*asset: Optional[str] = None*, *start_time: Optional[int] = None*, *end_time: Optional[int] = None*)

Return deposits stored in the database. Asset type and time filters can be used

Parameters

- **asset** (*Optional[str]*) – fetch only deposits of this asset
- **start_time** (*Optional[int]*) – fetch only deposits after this millistamp
- **end_time** (*Optional[int]*) – fetch only deposits before this millistamp

Returns The raw rows selected as saved in the database

Return type List[Tuple]

```
[
    ('azdf5e6a1d5z',    # transaction id
    1589479004000,      # deposit time
    'LTC',              # asset
    14.25),             # amount
]
```

get_spot_dividends(*asset: Optional[str] = None*, *start_time: Optional[int] = None*, *end_time: Optional[int] = None*)

Return dividends stored in the database. Asset type and time filters can be used

Parameters

- **asset** (*Optional[str]*) – fetch only dividends of this asset
- **start_time** (*Optional[int]*) – fetch only dividends after this millistamp
- **end_time** (*Optional[int]*) – fetch only dividends before this millistamp

Returns The raw rows selected as saved in the database

Return type List[Tuple]

```
[
    (8945138941,         # dividend id
    1594513589000,       # time
    'TRX',               # asset
    0.18745654),         # amount
]
```

get_spot_dusts(*asset: Optional[str] = None*, *start_time: Optional[int] = None*, *end_time: Optional[int] = None*)

Return dusts stored in the database. Asset type and time filters can be used

Parameters

- **asset** (*Optional[str]*) – fetch only dusts from this asset
- **start_time** (*Optional[int]*) – fetch only dusts after this millistamp
- **end_time** (*Optional[int]*) – fetch only dusts before this millistamp

Returns The raw rows selected as saved in the database

Return type List[Tuple]

```
[
    (82156485284,       # transaction id
    1605489113400,      # time
    'TRX',              # asset
    102.78415879,       # asset amount
    0.09084498,         # bnb amount
    0.00171514),        # bnb fee
]
```

get_spot_withdraws(*asset: Optional[str] = None*, *start_time: Optional[int] = None*, *end_time: Optional[int] = None*)

Return withdraws stored in the database. Asset type and time filters can be used

Parameters

- **asset** (*Optional[str]*) – fetch only withdraws of this asset
- **start_time** (*Optional[int]*) – fetch only withdraws after this millistamp
- **end_time** (*Optional[int]*) – fetch only withdraws before this millistamp

Returns The raw rows selected as saved in the database

Return type List[Tuple]

```
[
    ('84984dcqq5z11gyjfa',  # withdraw id
    'aazd8949vredqs56dz',   # transaction id
    1599138389000,          # withdraw time
    'XTZ',                  # asset
    57.0194,                # amount
    0.5),                   # fee
]
```

get_trades(*trade_type: str*, *start_time: Optional[int] = None*, *end_time: Optional[int] = None*, *asset: Optional[str] = None*, *ref_asset: Optional[str] = None*)

Return trades stored in the database. asset type, ref_asset type and time filters can be used

Parameters

- **trade_type** (*string, must be one of ('spot', 'cross_margin', 'isolated_margin')*) – type trade executed
- **start_time** (*Optional[int]*) – fetch only trades after this millistamp
- **end_time** (*Optional[int]*) – fetch only trades before this millistamp
- **asset** (*Optional[str]*) – fetch only trades with this asset
- **ref_asset** (*Optional[str]*) – fetch only trades with this ref_asset

Returns The raw rows selected as saved in the database

Return type List[Tuple]

Return for spot and cross margin:

```
[
    (384518832,         # trade_id
    1582892988052,      # trade time
    'BTC',              # asset
    'USDT',             # ref asset
    0.0015,             # asset quantity
```

(continues on next page)

(continued from previous page)

```
    9011.2,             # asset price to ref asset
    0.01425,            # fee
    'USDT',             # fee asset
    0),                 # is_buyer
]
```

Return for isolated margin:

```
[
    (384518832,         # trade_id
    1582892988052,      # trade time
    'BTCUSDT',          # symbol
    'BTC',              # asset
    'USDT',             # ref asset
    0.0015,             # asset quantity
    9011.2,             # asset price to ref asset
    0.01425,            # fee
    'USDT',             # fee asset
    0),                 # is_buyer
]
```

get_universal_transfers(*transfer_type: Optional[str] = None*, *asset: Optional[str] = None*, *start_time: Optional[int] = None*, *end_time: Optional[int] = None*)

Return universal transfers stored in the database. Transfer type, Asset type and time filters can be used

Parameters

- **transfer_type** (*Optional[str]*) – enum of the transfer type (ex: 'MAIN_MARGIN')
- **asset** (*Optional[str]*) – fetch only interests in this asset
- **start_time** (*Optional[int]*) – fetch only interests after this millistamp
- **end_time** (*Optional[int]*) – fetch only interests before this millistamp

Returns The raw rows selected as saved in the database

Return type List[Tuple]

```
[
    (1206491332,        # transfer id
    'MAIN_MARGIN',      # transfer type
    1589121841000,      # time
    'BNB',              # asset
    10.594112),         # amount
]
```

5.1.4 DataBase

class BinanceWatch.storage.DataBase.**DataBase**(*name: str*)

This class will be used to interact with sqlite3 databases without having to generates sqlite commands

__init__(*name: str*)

Initialise a DataBase instance

Parameters **name** (*str*) – name of the database

add_row(*table: BinanceWatch.storage.tables.Table*, *row: Tuple*, *auto_commit: bool = True*, *update_if_exists: bool = False*)

Add a row to a table

Parameters

- **table** (Table) – table to add a row to
- **row** (*Tuple*) – values to add to the database
- **auto_commit** (*bool*) – if the database state should be saved after the changes
- **update_if_exists** (*bool*) – if an integrity error is raised and this parameter is true, will update the existing row

Returns None

Return type None

add_rows(*table: BinanceWatch.storage.tables.Table*, *rows: List[Tuple]*, *auto_commit: bool = True*, *update_if_exists: bool = False*)

Add several rows to a table

Parameters

- **table** (Table) – table to add a row to
- **rows** (*List[Tuple]*) – list of values to add to the database
- **auto_commit** (*bool*) – if the database state should be saved after the changes
- **update_if_exists** (*bool*) – if an integrity error is raised and this parameter is true, will update the existing row

Returns None

Return type None

close()

Close the connection with the sqlite3 database

Returns None

Return type None

commit()

Submit and save the database state

Returns None

Return type None

connect()

Connect to the sqlite3 database

Returns None

Return type None

create_table(*table:* BinanceWatch.storage.tables.Table)
Create a table in the database

Parameters **table** (Table) – Table instance with the config of the table to create

Returns None

Return type None

drop_all_tables()
Drop all the tables existing in the database

Returns None

Return type None

drop_table(*table: Union[*BinanceWatch.storage.tables.Table*, str]*)
Delete a table from the database

Parameters **table** (*Union[Table, str]*) – table or table name to drop

Returns None

Return type None

get_all_rows(*table:* BinanceWatch.storage.tables.Table) → List[Tuple]
Get all the rows of a table

Parameters **table** (Table) – table to get the rows from

Returns all the rows of the table

Return type List[Tuple]

get_all_tables() → List[Tuple]
Return all the tables existing in the database

Returns tables descriptions

Return type List[Tuple]

get_conditions_rows(*table:* BinanceWatch.storage.tables.Table, *selection: Union[str, List[str]] = '**', *conditions_list: Optional[List[Tuple[str,* BinanceWatch.storage.DataBase.SQLConditionEnum*, Any]]] = None*, *order_list: Optional[List[str]] = None*) → List[Tuple]
Select rows with optional conditions and optional order

Parameters

- **table** (Table) – table to select the rows from
- **selection** (*Union[str, List[str]]*) – list of column or SQL type selection
- **conditions_list** (*Optional[List[Tuple[str,* SQLConditionEnum*, Any]]]*) – list of conditions to select the row
- **order_list** (*Optional[List[str]]*) – List of SQL type order by

Returns the selected rows

Return type List[Tuple]

static get_create_cmd(*table:* BinanceWatch.storage.tables.Table) → str
Return the command in string format to create a table in the database

Parameters **table** (Table) – Table instance with the config if the table to create

Returns execution command for the table creation

Return type str

get_row_by_key(*table:* BinanceWatch.storage.tables.Table, *key_value*) → Optional[Tuple]
Get the row identified by a primary key value from a table

Parameters

- **table** (*Table*) – table to fetch the key from
- **key_value** (*Any*) – value of the primary key

Returns the raw row of of the table

Return type Optional[Tuple]

update_row(*table:* BinanceWatch.storage.tables.Table, *row: Tuple*, *auto_commit=True*)
Update the value of a row in a table

Parameters

- **table** (*Table*) – table to get updated
- **row** (*Tuple*) – values to update
- **auto_commit** (*bool*) – if the database state should be saved after the changes

Returns None

Return type None

class BinanceWatch.storage.DataBase.**SQLConditionEnum**(*value*)
Enumeration for SQL comparison operator

https://www.techonthenet.com/sqlite/comparison_operators.php

diff = '!='

equal = '='

greater = '>'

greater_equal = '>='

lower = '<'

lower_equal = '<='

5.1.5 Tables

class BinanceWatch.storage.tables.**Table**(*name: str*, *columns_names: List[str]*, *columns_sql_types: List[str]*, *primary_key: Optional[str] = None*, *primary_key_sql_type: Optional[str] = None*)
This class represent a table in a database. All columns names are dynamic attributes @DynamicAttrs This class is used to describe the tables that will be used to in the database

__init__(*name: str*, *columns_names: List[str]*, *columns_sql_types: List[str]*, *primary_key: Optional[str] = None*, *primary_key_sql_type: Optional[str] = None*)
Initialise a Table instance

Parameters

- **name** (*str*) – name of the table
- **columns_names** (*List[str]*) – names of the columns (except primary column)

- **columns_sql_types** (*List[str]*) – sql types of the previous columns
- **primary_key** (*Optional[str]*) – name of the primary key (None, if no primary key is needed)
- **primary_key_sql_type** (*Optional[str]*) – sql type of the primary key (None, if no primary key is needed)

5.2 Indices and tables

- genindex
- modindex
- search

PYTHON MODULE INDEX

b

INDEX

Symbols

A

B

C