BTCTURK API - PYTHON

Release 1.6.0

Mirac Baydemir

Feb 17, 2021

CONTENTS:

CHAPTER

ONE

QUICKSTART

btcturkapi-python is a wrapper library built around Btcturk's REST api implementation.

It provides more abstract version of REST Client for traders using Btcturk as a cryptocurrency platform.

1.1 Installation

You can install this library via pip.

It is the best practice using a virtual environment for your project, keep that in mind.

Run this command through terminal:

```
$ pip install btcturk-api
```

Since the project's current version doesn't have support for websockets and any advanced features, dependencies are simple and you should not encounter any installation error.

1.2 Usage

After installing the library, just import the client and you are good to go. You can use any feature btcturk api provides without dealing with preparing requests, handling responses etc.

```
>>> from btcturk_api.client import Client
```

You can use public endpoints without providing an api key/secret pair.

```
>>> my_client = Client()
>>> my_client.get_server_time()
{'serverTime': 1613292018131, 'serverTime2': '2021-02-14T08:40:18.1308832+00:00'}
```

If you have an api key/secret pair, you can use *account endpoints* and *trading operations*

```
>>> my_client = Client(api_key='<Your Api Key>', api_secret='<Your Api Secret>')
>>> my_client.get_account_balance()
[
  {
   'asset': 'TRY',
   'assetname': 'Türk Lirası',
   'balance': '0.8462753436459292',
   'locked': '0',
```

(continues on next page)

(continued from previous page)

```
    'free': '0.8462753436459292',
    'orderFund': '0',
    'requestFund': '0',
    'precision': 2
    },
    {...}
]
```

CHAPTER

TWO

PUBLIC ENDPOINTS

2.1 Get Server Time

If you are encountering BTCTurkAuthenticationError, reason might be time inconsistency between server and your machine. You can get server's time with this method:

```
>>> my_client.get_server_time()
{
  'serverTime': 1613313462021,
  'serverTime2': '2021-02-14T14:37:42.0214779+00:00'
}
```

2.2 Get Exchange Info

```
>>> my_client.get_exchange_info()
[
  {
    'id': 1,
    'name': 'BTCTRY',
    'nameNormalized': 'BTC_TRY',
    'status': 'TRADING',
    'numerator': 'BTC',
    'denominator': 'TRY',
    'numeratorScale': 8,
    'denominatorScale': 2,
    'hasFraction': False,
    'filters': [{'filterType': 'PRICE_FILTER', 'minPrice': '0.0000000000001',
↪'maxPrice': '10000000', ....],
    'orderMethods': ['MARKET', 'LIMIT', 'STOP_MARKET', 'STOP_LIMIT'],
    'displayFormat': '#,###',
    'commissionFromNumerator': False,
    'order': 1000,
    'priceRounding': False
  },
  {...}
]
```

2.3 Get Ticker

```
>>> my_client.tick()
{
  'pair': 'BTCTRY',
  'pairNormalized': 'BTC_TRY',
  'timestamp': 1613313834273,
  'last': 350000.0,
  'high': 354975.0,
  'low': 332565.0,
  'bid': 350000.0,
  'ask': 350904.0,
  'open': 332775.0,
  'volume': 1718.94206874,
  'average': 344569.69406522,
  'daily': 18129.0,
  'dailyPercent': 5.18,
  'denominatorSymbol': 'TRY',
  'numeratorSymbol': 'BTC',
  'order': 1000
  }
```

2.4 Get Order book

```
>>> my_client.get_order_book(pair='BTCTRY', limit=1)
{'timestamp': 1613315997466.0,
'bids': [['349600.00', '0.00518551']],
'asks': [['349830.00', '10.62911645']]
}
```

2.5 Get Trades

```
>>> my_client.get_trades(pair='BTCTRY')
[
  {
    'pair': 'BTCTRY',
    'pairNormalized': 'BTC_TRY',
    'numerator': 'BTC',
    'denominator': 'TRY',
    'date': 1613316100877,
    'tid': '637489129008759423',
    'price': '349000.00',
    'amount': '0.00500000',
    'side': 'sell'
   },
   {....}
]
```

CHAPTER

THREE

ACCOUNT RELATED ENDPOINTS

3.1 Get Account Balance

You can use this method for accessing your account balance information.

Method returns all balances by default, if you want specific assets, you should give list of assets as 'assets' parameter.

Here's an example, returning BTC and ETH balance information:

```
>>> my_client.get_account_balance(assets=['BTC', 'ETH'])
[
  {
    'asset': 'BTC',
    'assetname': 'Bitcoin',
    'balance': '0.0003670154826363',
    'locked': '0',
    'free': '0.0003670154826363',
    'orderFund': '0',
    'requestFund': '0',
    'precision': 8
  },
  {
    'asset': 'ETH',
    'assetname': 'Ethereum',
    'balance': '0',
    'locked': '0',
    'free': '0',
    'orderFund': '0',
    'requestFund': '0',
    'precision': 8
  }
]
```

If you want to know more about this response and method's usage, check detailed information by clicking here.

3.2 Get Trade History

Usage Scenerio: Users want to access their trade history for last **90** days. But only **buy** trades needed and they want only bitcoin and ripple trades.

```
>>> from datetime import datetime, timedelta
>>> start_date = (datetime.now() - timedelta(days=90)).timestamp()
>>> start_date = int(start_date * 1000)  # Timestamp should be in milliseconds
>>> my_client.get_trade_history(
        trade_type=['buy'],
        symbol=['btc', 'xrp'],
        start_date=start_date
    )
```

3.3 Get Crypto History

Scenerio: Users want to access their trade history for last **90** days. But only **buy** orders needed and they want only bitcoin and ripple trades.

```
>>> from datetime import datetime, timedelta
>>> start_date = (datetime.now() - timedelta(days=90)).timestamp()
>>> start_date = int(start_date * 1000)  # Timestamp should be in milliseconds
>>> my_client.get_trade_history(
        trade_type=['buy'],
        symbol=['btc', 'xrp'],
        start_date=start_date
    )
```

3.4 Get Fiat History

Scenerio: Users want to access their trade history for last **90** days. But only **buy** orders needed and they want only bitcoin and ripple trades.

```
>>> from datetime import datetime, timedelta
>>> start_date = (datetime.now() - timedelta(days=90)).timestamp()
>>> start_date = int(start_date * 1000)  # Timestamp should be in milliseconds
>>> my_client.get_trade_history(
        trade_type=['buy'],
        symbol=['btc', 'xrp'],
        start_date=start_date
    )
```

3.5 Get Open Orders

Usage Scenario: Users want to get information about their open orders for all cryptocurrencies

```
>>> my_client.get_open_orders()
```

3.6 Get All Orders

Usage Scenario: Users want to get their all ripple orders for last **6 hours**:

```
>>> from datetime import datetime, timedelta
>>> start_date = (datetime.now() - timedelta(hours=6)).timestamp()
>>> start_date = int(start_date * 1000)  # Timestamp should be in milliseconds
>>> my_client.get_all_orders(
        pair_symbol='XRP_TRY',
        start_date=start_date
    )
```

CHAPTER

FOUR

TRADE OPERATIONS

4.1 Important Note About Orders

We are going to talk about concepts like quantity, price in next subsections of this chapter.

Users usually misinterpret those concepts and get various scaling errors, let's dive in these concepts one by one.

4.1.1 Numerator / Denominator

These concepts will be used to explain trades in a more clear way. In a trade pair such as **BTC/USDT**, first one is Numerator, second one is Denominator. In this example Numerator is **BTC**, Denominator is **USDT**.

4.1.2 Quantity

In market orders, since price is determined by current market value you only provide a **quantity parameter** for your trade.

If you are **buying**, denominator will be used as quantity unit, but if you are selling numerator becomes the quantity.

If you are confused with these terms, don't worry! Here's a few cases which includes might want to sell or buy bitcoin. After looking these cases, you will understand why people misinterprets the concept quantity and what actually means.

Quick Example

I want to trade Bitcoin with USDT.

Case 1:

I want to buy bitcoin.

Wrong Approach:

I want to buy 0.05 bitcoin, so this is my quantity. I can call submit market order method with 'quantity=0.05'

WRONG! This is misinterpreted by beginner client users all the time.

If you provide 0.05 as quantity for BTC/USDT pair, **you tell the api that you want to buy bitcoin for worth of 0.05 USDT**, you will get either min_total_value error or scaling error if you do that.

Don't forget, since you are buying, denominator in this pair should be your quantity. Thus, USDT!

Correct Approach:

I'm going to buy Bitcoin with my USDT with market price, i want to trade my 100 USDT with bitcoin, so my quantity is 100.

Case 2:

I want to sell bitcoin.

Selling should not be confusing. In BTC/USDT pair, if you want to sell 0.05 Bitcoin, your quantity is 0.05. Pretty straightforward

4.1.3 Price

Price is pretty straightforward too. It is the value of cryptocurrency in stock exchange.

You can only use price parameter with stop orders and limit orders.

4.2 Submit Market Order / Code Examples

Usage Example 1:

- Pair: XRP/USDT
- Goal: Buying XRP for 100 USDT

```
>>> my_client.submit_market_order(
      quantity=100.0,
      order_type='buy',
      pair_symbol='XRP_USDT'
    )
```

Usage Example 2:

- Pair: ETH/TRY
- Goal: Selling 1250 ETH exchange of TRY

```
>>> my_client.submit_market_order(
      quantity=1250.0,
      order_type='sell',
      pair_symbol='ETH_TRY'
    )
```

4.3 Submit Limit Order / Code Examples

Usage Example 1:

- Pair: XRP/USDT
- Goal: Place a Limit Buy Order for 400 XRP with price of 0.16 USDT

```
>>> my_client.submit_limit_order(
      quantity=400.0,
      price=0.16,
      order_type='buy',
      pair_symbol='XRP_USDT'
    )
```

Usage Example 2:

- Pair: ETH/TRY

- Goal: Place a Limit Sell Order for 1250 ETH with price of 1950 USDT

```
>>> my_client.submit_limit_order(
      quantity=1250.0,
      price=1950,
      order_type='sell',
      pair_symbol='ETH_USDT'
    )
```

4.4 Submit Stop Limit Order / Code Examples

Usage Example 1:

- Pair: BTC/USDT
- Goal: If Bitcoin price hits 50.000 USDT, we're going to place a limit buy order with quantity=0.05 and price=50.500

```
>>> my_client.submit_limit_order(
      quantity=0.05,
      price=50500,
      stop_price=50000
      order_type='buy',
      pair_symbol='BTC_USDT'
    )
```

Usage Example 2:

- Pair: BTC/USDT
- Goal: If Bitcoin price drops 40.000 USDT, we're going to place a limit sell order with quantity=0.05 and price=39.500

```
>>> my_client.submit_limit_order(
      quantity=0.05,
      price=39500,
      stop_price=40000
      order_type='sell',
      pair_symbol='BTC_USDT'
    )
```

CHAPTER

FIVE

EXCEPTIONS

5.1 InvalidRequestParameterError

When you provide invalid parameters, you get this exception. For proper usage of parameters, check BTCTurk API section of this documentation and learn more about the method you want to use.

5.2 BTCTurkAuthenticationError

When server can't authenticate you, this exception will be raised. In Exceptions sections, reasons of that explained.

5.3 RequestLimitExceededError

When you exceed Btcturk's api response limits, this exception will be raised. Check Btcturk's documentation for api limits.

5.4 InternalServerError

Raised for 5xx errors. If response message is html instead of json.

CHAPTER

SIX

COMMON ERRORS & SOLUTIONS

In this section, you will find common errors that you may encounter while using the client.

6.1 BTCTurkAuthenticationError

One of the most common errors that you may encounter is BTCTurkAuthenticationError which means Server has unable to recognize your identity. Thus, you can't authenticate.

6.1.1 Wrong API Credentials

- When you are creating your key/secret pair, you provide your machine's public IPv4 address. Make sure that you are providing the correct address. In addition to that, if you are using this client on a Virtual Machine, you must get credentials for that machine's public IPv4 address, always keep that in mind.
- When you are doing copy/paste, you may miss some characters, always check when you paste your credentials.
- Make sure you have authorized your credentials with correct permissions (buy-sell / see balance etc.)

6.1.2 Invalid Nonce

In order to authenticate, your machine's time and Btcturk Server's time must match. If it doesn't, you will get an authentication error with Invalid Nonce message. Again, if you are using a virtual machine, make sure these times match.

You can check Btcturk's server time by creating a client without key/secret pair and calling *client.get_server_time()* method.

CHAPTER

SEVEN

BTCTURK API REFERENCE

7.1 btcturk_api.client module

class btcturk_api.client.**Client**(*api_key: Optional[str] = None*, *api_secret: Optional[str] = None*)

Bases: object

API Client Class

Methods

get_exchange_info:	Method for getting exchange info of any given pair
get_server_time:	Gets Current Server Time
tick:	Gets price related information of any given pair
get_order_book:	Gets the order book of given pair
get_trades:	Gets a list of Trades for given pair
get_account_balance:	Gets the list of balances which user have
get_trade_history:	Gets the history of user's trades.
get_crypto_history:	Gets the history of user's crypto transactions.
get_fiat_history:	Gets the history of user's fiat transactions.
get_open_orders:	Get's list of user's open orders for given pair
get_all_orders:	Get's users all orders for given pair
cancel_order:	Deletes The Order
submit_market_order:	Submits an order in type of 'market order'
submit_limit_order:	Submits an order in type of 'limit order'
submit_stop_order:	Submits an order in type of 'stop order'

API_BASE = 'https://api.btcturk.com'

API_ENDPOINT_AUTH = '/api/v1/'

API_ENDPOINT_NON_AUTH = '/api/v2/'

API_ENDPOINT_TRANSACTIONS = '/api/v1/users/transactions/'

cancel_order(*order_id=None*)

Deletes The Order

Parameters

order_id [int, mandatory]

Returns

bool Success value if there is no exception raised by handler

get_account_balance(*assets=None*)

Gets the list of balances which user have

Parameters

assets: optional List of assets like [‘BTC’, ‘TRY’, . . .]

Returns

list Example of Response Format

```
[
  {
    'asset': 'EUR',
    'assetname': 'Euro',
    'balance': '0',
    'locked': '0',
    'free': '0',
    'orderFund': '0',
    'requestFund': '0',
    'precision': 2
  },
  ...
]
```

get_all_orders(*order_id: int = 0*, *pair_symbol=None*, *start_date=None*, *end_date=None*, *page=None*, *limit=100*, ***kwargs*)

Get’s users all orders for given pair

If you specify kwargs, the other parameters will be **overridden**. Only keyword arguments you specified will be used to construct a query.

Therefore, it is your choice to use kwargs.

But i strongly discourage you to use that for avoiding any invalid requests

If start_date not specified, it will get orders for last 30 days.

Parameters

order_id: int, optional If orderId set, it will return all orders greater than or equals to this order id

pair_symbol: str, mandatory Like BTC_TRY, XRP_USDT..

start_date: int, optional start date as timestamp in milliseconds

end_date: int, optional end date as timestamp in milliseconds

page: int, optional page number

limit: int, optional limit number

kwargs

Returns

list List of data dictionaries

Example Of Response Format

```
[
  {
    'id': '<Order id>',
    'price': '<Price of the order>',
    'amount': '<Amount of the order>',
    'quantity': '<Quantity of the order>',
    'pairsymbol': '<Pair of the order>',
    'pairSymbolNormalized': '<Pair of the order with "_" in␣
↪between>',
    'type': '<Type of order. Buy or Sell>',
    'method': '<Method of order. Limit, Stop Limit..>',
    'orderClientId': '<Order client id created with>',
    'time': '<Unix time the order was inserted at>',
    'updateTime': '<Unix time last updated>',
    'status': '<Status of the order. Untouched, Partial..>',

  },
  ...
]
```

get_crypto_history(*symbol=None*, *transaction_type=None*, *start_date=None*, *end_date=None*, ***kwargs*)

Gets the history of user's crypto transactions.

If symbol not specified, all crypto symbols will be used

If transaction_type not specified, both 'withdrawal' and 'deposit' types will be used

If start_date not specified, it will get trades for last 30 days.

If you specify kwargs, the other parameters will be **overridden**. Only keyword arguments you specified will be used to construct a query. Therefore, it is your choice to use kwargs.

But i strongly discourage you to use that for avoiding any invalid requests

Parameters

symbol [list, optional] ["btc", "try", ... etc.]

transaction_type [list , optional] ["deposit", "withdrawal"], ["deposit"] or ["withdrawal"]

start_date [timestamp, optional]

end_date [timestamp, optional]

kwargs

Returns

list List of trade data dictionaries,

Example of Response Format

```
[
  {
    'balanceType': '<Type of transaction (deposit, withdrawal)>',
    'currencySymbol': '<Transaction currency symbol>',
    'id': '<Transaction id>',
    'timestamp': '<Unix timestamp>',
    'funds': '<Funds>',
    'orderFund': '<Transaction Amount>',
    'fee': '<Transaction fee>',
```

(continues on next page)

(continued from previous page)

```
    'tax': <Transaction tax>
  },
  ...
]
```

get_exchange_info(*symbol_list=None*)

Method for getting exchange info of any given pair

Parameters

symbol_list [list, optional] In format of [‘BTCUSDT’, ‘XRPUSDT’, ‘ETHTRY’ . . .]

Returns

list If symbol_list is None, list of data dictionaries of all pairs. Otherwise, that list filtered down to given symbol list

Example of Response Format

```
[
  {
    'id': '',
    'name': '',
    'nameNormalized': '',
    'status': '',
    'numerator': '',
    'denominator': '',
    'numeratorScale': '',
    'denominatorScale': '',
    'hasFraction': '',
    'filters': ''
    'orderMethods': ['MARKET', 'LIMIT', 'STOP_MARKET', 'STOP_LIMIT
↪'], # IMPORTANT
    'displayFormat': '#,###',
    'commissionFromNumerator': False, 'order': 999,
    'priceRounding': False
  },
  ...
]
```

get_fiat_history(*balance_types=None*, *currency_symbols=None*, *start_date=None*, *end_date=None*, ***kwargs*)

Gets the history of user’s fiat transactions.

If balance_types not specified, both ‘withdrawal’ and ‘deposit’ types will be used

If currency_symbols not specified, all currency symbols will be used

If start_date not specified, it will get trades for last 30 days.

If you specify kwargs, the other parameters will be **overridden**. Only keyword arguments you specified will be used to construct a query. Therefore, it is your choice to use kwargs.

But i strongly discourage you to use that for avoiding any invalid requests

Parameters

balance_types [list, optional] [“buy”, “sell”]

currency_symbols [list, optional] [“try”, . . . etc.]

start_date [timestamp, optional]

end_date [timestamp, optional]

kwargs

Returns

list List of trade data dictionaries,

Example of Response Format

```
[
  {
    'balanceType': '<Type of transaction (deposit, withdrawal)>',
    'currencySymbol': '<Transaction currency symbol>',
    'id': '<Transaction id>',
    'timestamp': '<Unix timestamp>',
    'funds': '<Funds>',
    'orderFund': '<Transaction Amount>',
    'fee': '<Transaction fee>',
    'tax': <Transaction tax>
  },
  ...
]
```

get_open_orders (*pair=None*, ***kwargs*)

Get's list of user's open orders for given pair

If you specify kwargs, the other parameters will be **overridden**. Only keyword arguments you specified will be used to construct a query. Therefore, it is your choice to use kwargs.

But i strongly discourage you to use that for avoiding any invalid requests

Parameters

pair [str, optional] if not set returns all pairs open orders

kwargs

Returns

dict Data dictionary

Example of Response Format

```
{
  'id': '<Order id>',
  'price': '<Price of the order>',
  'amount': '<Quantity of the order>',
  'pairsymbol': '<Pair of the order>',
  'pairSymbolNormalized': '<Pair of the order with "_" in between.
↪>',
  'type': '<Type of order. Buy or Sell>',
  'method': '<Method of order. Limit, Stop Limit..>',
  'orderClientId': <Order client id created with>,
  'time': '<Unix time the order was inserted at>',
  'updateTime': '<Unix time last updated>',
  'status': <Status of the order. Untouched, Partial>
},
```

get_order_book (*pair=None*, *limit=100*, ***kwargs*)

Gets the order book of given pair

If you specify kwargs, the other parameters will be **overridden**. Only keyword arguments you specified will be used to construct a query. Therefore, it is your choice to use kwargs.

But i strongly discourage you to use that for avoiding any invalid requests

Parameters

pair [str, mandatory] pair symbol like 'BTC_TRY', 'ETH_BTC', ...

limit [int, optional] default 100 max 1000

kwargs

Returns

dict data dictionary

Example of Response Format

```
[
  {
    'timestamp': '<Current Unix time in milliseconds>',
    'bids': '<Array of current open bids on the orderbook>',
    'asks': '<Array of current open asks on the orderbook>',
  },
  ...
]
```

get_server_time()

Gets Current Server Time

Returns

dictionary Example of Response Format

```
{
    'serverTime': '<Unix Timestamp as int>',
    'serverTime2': '<Datetime string>',
}
```

get_trade_history(*trade_type=None*, *symbol=None*, *start_date=None*, *end_date=None*, ***kwargs*)

Gets the history of user's trades.

If trade_type not specified, both 'buy' and 'sell' types will be used

If symbol not specified, all crypto symbols will be used

If start_date not specified, it will get trades for last 30 days.

If you specify kwargs, the other parameters will be **overridden.** Only keyword arguments you specified will be used to construct a query. Therefore, it is your choice to use kwargs.

But i strongly discourage you to use that for avoiding any invalid requests

Parameters

trade_type [list, optional] ["buy", "sell"], ["buy"] or ["sell"]

symbol [list -> str, optional] ["btc", "try", ... etc.],

start_date [timestamp, optional]

end_date [timestamp, optional]

kwargs

Returns

list List of trade data dictionaries,

Example of Response Format

```
[
  {
    'price': '<Trade price>',
    'numeratorSymbol': '<Trade pair numerator symbol>',
    'denominatorSymbol': '<Trade pair denominator symbol>',
    'orderType': '<Trade type (buy,sell)>',
    'id': '<Trade id>',
    'timestamp': '<Unix timestamp>',
    'amount': '<Trade Amount (always negative if order type is␣
↪sell)>',
    'fee': '<Trade fee>',
    'tax': '<Trade tax>'
  },
  ...
]
```

get_trades(*pair=None*, *last=50*, ***kwargs*)

Gets a list of Trades for given pair

If you specify kwargs, the other parameters will be **overridden.** Only keyword arguments you specified will be used to construct a query. Therefore, it is your choice to use kwargs.

But i strongly discourage you to use that for avoiding any invalid requests

Parameters

pair [str, mandatory] pair symbol like 'BTC_TRY', 'ETH_BTC'..

last [int, optional] default 50 max 1000

Returns

dict Data dictionary

Example of Response Format

```
{
    'pair': '<Requested pair symbol>',
    'pairNormalized': '<Request Pair symbol with "_" in between.>
↪',
    'numerator': '<Numerator currency for the requested pair>',
    'denominator': '<Denominator currency for the requested pair>
↪',
    'date': '<Unix time of the trade in milliseconds>',
    'tid': '<Trade ID>',
    'price': '<Price of the trade>',
    'amount': '<Amount of the trade>',
},
```

submit_limit_order(*quantity=0.0*, *price=0.0*, *order_type=None*, *pair_symbol=None*, *new_order_client_id=None*)

Submits an order in type of 'limit order'

Parameters

quantity [decimal, mandatory] Mandatory for market or limit orders.

price [decimal, mandatory] Price field will be ignored for market orders.

order_type [str, mandatory] 'buy' or 'sell'

pair_symbol [str, mandatory] Like 'BTC_TRY', 'XRP_USDT'..

new_order_client_id [str, optional]

Returns

dict Dictionary of order information

Example of Response Format

```
{
  'id': '<order id>',
  'datetime': '<timestamp>',
  'type': '<Buy or sell>',
  'method': '<method of order (limit,stop,market)>',
  'price': '<price>',
  'stopPrice': '<stop price>',
  'quantity': '<quantity>',
  'pairSymbol': '<pair symbol>',
  'pairSymbolNormalized': '<normalized pair symbol>',
  'newOrderClientId': '<guid>',
},
```

Raises

ValueError If wrong pair_symbol entered

submit_market_order(*quantity=0.0*, *order_type=None*, *pair_symbol=None*, *new_order_client_id=None*)

Submits an order in type of 'market order'

Parameters

quantity [decimal, mandatory] Mandatory for market or limit orders.

order_type [str, mandatory] 'buy' or 'sell'

pair_symbol [str, mandatory]

new_order_client_id [str, optional]

Returns

dict Dictionary of order information

Example of Response Format

```
{
  'id': '<order id>',
  'datetime': '<timestamp>',
  'type': '<Buy or sell>',
  'method': '<method of order (limit,stop,market)>',
  'price': '<price>',
  'stopPrice': '<stop price>',
  'quantity': '<quantity>',
  'pairSymbol': '<pair symbol>',
  'pairSymbolNormalized': '<normalized pair symbol>',
  'newOrderClientId': '<guid>',
},
```

Raises

ValueError If wrong pair_symbol entered or file cache for scales hasn't been updated

submit_stop_order(*stop_price=0.0*, *quantity=0.0*, *price=0.0*, *order_type=None*, *order_method=None*, *pair_symbol=None*, *new_order_client_id=None*)

Submits an order in type of 'stop order'

Parameters

stop_price: decimal, mandatory For stop orders

quantity [decimal, mandatory] Mandatory for market or limit orders.

price [decimal, mandatory] Price field will be ignored for market orders.

order_type [str, mandatory] 'buy' or 'sell'

order_method: str, mandatory Either 'stopLimit' or 'stopMarket'

pair_symbol [str, mandatory]

new_order_client_id [str, optional]

Returns

dict Dictionary of order information

Example of Response Format

```
{
  'id': '<order id>',
  'datetime': '<timestamp>',
  'type': '<Buy or sell>',
  'method': '<method of order (limit,stop,market)>',
  'price': '<price>',
  'stopPrice': '<stop price>',
  'quantity': '<quantity>',
  'pairSymbol': '<pair symbol>',
  'pairSymbolNormalized': '<normalized pair symbol>',
  'newOrderClientId': '<guid>',
},
```

Raises

ValueError If wrong pair_symbol entered

tick(*pair=None*, ***kwargs*)

Gets price related information of any given pair

If you specify kwargs, the other parameters will be **overridden.** Only keyword arguments you specified will be used to construct a query. Therefore, it is your choice to use kwargs.

But i strongly discourage you to use that for avoiding any invalid requests

Parameters

pair [str, optional] pair symbol like 'BTC_TRY', 'ETH_BTC', ...

kwargs

Returns

list If pair is set, a list of data dictionary with given pair, (length=1) Otherwise, a list of data dictionaries of all pairs.

Example of Response Format

```
[
  {
    'pairSymbol': '<Requested pair symbol>',
    'pairSymbolNormalized': '<Requested pair symbol with "_" in␣
↪between.>',
    'timestamp': '<Current Unix time in milliseconds>'
    'last': '<Last price>',
    'high': '<Highest trade price in last 24 hours>',
    'low': '<Lowest trade price in last 24 hours>',
    'bid': '<Highest current bid>',
    'ask': '<Lowest current ask>',
    'open': '<Price of the opening trade in last 24 hours>',
    'volume': '<Total volume in last 24 hours>',
    'average': '<Average Price in last 24 hours>',
    'daily': '<Price change in last 24 hours>',
    'dailyPercent': '<Price change percent in last 24 hours>',
    'denominatorSymbol': '<Denominator currency symbol of the␣
↪pair>',
    'numeratorSymbol': '<Numerator currency symbol of the pair>',
  },
  ...
]
```

CHAPTER

EIGHT

INDICES AND TABLES

- genindex

PYTHON MODULE INDEX

b

INDEX