1 Overview 3
  1.1 Supported stocks ................................................................. 3

2 Instalation 5

3 Configuration 7

4 Warnings 9

5 Constants 11

6 Ticker 13
  6.1 Example of using ................................................................. 13
  6.2 Stock argument ................................................................. 14
  6.3 Currencies arguments ....................................................... 14
  6.4 Invalid ................................................................. 14
  6.5 Class Ticker ................................................................. 14
  6.6 Mapping ................................................................. 14

7 Trades 15

8 Orderbook 17

9 General informations 19
  9.1 Handeling exceptions ....................................................... 19
  9.2 How it wokrs ................................................................. 19
  9.3 Response validation ....................................................... 19
  9.4 Congestion control ....................................................... 20

10 Bifinex 21
  10.1 Fundingbook ................................................................. 21
  10.2 Lends ................................................................. 23
  10.3 Orderbook ................................................................. 24
  10.4 Stats ................................................................. 26
  10.5 Symbols ................................................................. 27
  10.6 Symbols details ............................................................. 28
  10.7 Ticker ................................................................. 29
  10.8 Trades ................................................................. 30
## Bitstamp
- 11.1 EUR USD conversion rate ................................................................. 33
- 11.2 Hourly ticker ....................................................................................... 34
- 11.3 Orderbook ............................................................................................ 35
- 11.4 Ticker .................................................................................................... 36
- 11.5 Transactions ......................................................................................... 37

## Bittrex
- 12.1 Get currencies ...................................................................................... 39
- 12.2 Get markets .......................................................................................... 40
- 12.3 Get market history ................................................................................ 42
- 12.4 Get market summary ............................................................................ 43
- 12.5 Get market summaries ......................................................................... 44
- 12.6 Get orderbook ........................................................................................ 46
- 12.7 Get ticker ................................................................................................ 47

## Btcc

## Btccpro

## Btccusd

## Btce
- 16.1 Depth .................................................................................................. 63
- 16.2 Info ...................................................................................................... 64
- 16.3 Ticker .................................................................................................... 65
- 16.4 Trades ................................................................................................... 66

## Bter
- 17.1 Depth .................................................................................................. 69
- 17.2 Market info .......................................................................................... 70
- 17.3 Market details ....................................................................................... 71
- 17.4 Ticker .................................................................................................... 73
- 17.5 Tickers ................................................................................................... 74
- 17.6 Trading pairs ......................................................................................... 76
- 17.7 Trade history ......................................................................................... 76

## Cexio
- 18.1 Chart .................................................................................................... 79
- 18.2 Convert .................................................................................................. 80
- 18.3 Currency limits ...................................................................................... 81
- 18.4 Historical 1m OHLCV chart ................................................................. 82
- 18.5 Last price ............................................................................................... 83
- 18.6 Last prices for given market .................................................................. 84
- 18.7 Orderbook ............................................................................................. 85
- 18.8 Ticker .................................................................................................... 87
- 18.9 Tickers for all pairs by market ............................................................... 88
- 18.10 Trade history ....................................................................................... 89

## Kraken
- 19.1 Asset info .............................................................................................. 91
- 19.2 Asset pairs ............................................................................................ 93
- 19.3 OHLC .................................................................................................... 95
- 19.4 Orderbook ............................................................................................. 96
- 19.5 Ticker .................................................................................................... 98
CCS is Python package for communication with stocks which is trading with crypto currencies.
Overview

Crypto currencies stocks (ccs) is Python package for communication with stocks which is trading with crypto currencies. This library has two levels:

- basic stock’s API
- unificated API build over basic API

It means here are two ways how use this library. using basic api and than parse himself using unificated api

Public api

REST implemented method

Here is effort solve problem that crypto currencies stocks offer similarly information in different formats. Unification is associated with:

- API unification for most frequented requests (ticker, trade history, orderbook ...)
- responses unification from requests

### 1.1 Supported stocks

<table>
<thead>
<tr>
<th>Stock</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bitfinex</td>
<td><a href="https://www.bitfinex.com/">https://www.bitfinex.com/</a></td>
</tr>
<tr>
<td>Bitstamp</td>
<td><a href="https://www.bitstamp.net/">https://www.bitstamp.net/</a></td>
</tr>
<tr>
<td>Bittrex</td>
<td><a href="https://bittrex.com/">https://bittrex.com/</a></td>
</tr>
<tr>
<td>Btc</td>
<td><a href="https://www.btc.com/">https://www.btc.com/</a></td>
</tr>
<tr>
<td>Btce</td>
<td><a href="https://btc-e.com/">https://btc-e.com/</a></td>
</tr>
<tr>
<td>Bter</td>
<td><a href="https://bter.com/">https://bter.com/</a></td>
</tr>
<tr>
<td>Cex.io</td>
<td><a href="https://cex.io/">https://cex.io/</a></td>
</tr>
<tr>
<td>Kraken</td>
<td><a href="https://www.kraken.com/">https://www.kraken.com/</a></td>
</tr>
<tr>
<td>Okcoin.com</td>
<td><a href="https://www.okcoin.com/">https://www.okcoin.com/</a></td>
</tr>
<tr>
<td>Okcoin.cn</td>
<td><a href="https://www.okcoin.cn/">https://www.okcoin.cn/</a></td>
</tr>
<tr>
<td>Poloniex</td>
<td><a href="https://poloniex.com/">https://poloniex.com/</a></td>
</tr>
</tbody>
</table>
This package is part of pypi. Command for installation is:

```
$ pip install ccs
```
CHAPTER 3

Configuration
CHAPTER 5

Constants
6.1 Example of using

```python
>>> import ccs
ticker = ccs.ticker(ccs.constants.BITFINEX, ccs.constants.BTC, ccs.constants.USD)
>>> print(str(ticker))
... >>> print(str(ticker.usymbol()))
... >>> print(str(ticker.osymbol()))
... >>> print(str(ticker.stock()))
... >>> print(str(ticker.last()))
... >>> print(str(ticker.low()))
... >>> print(str(ticker.high()))
... >>> print(str(ticker.ask()))
... >>> print(str(ticker.bid()))
... >>> print(str(ticker.volume24h()))
... >>> print(str(ticker.timestamp()))
... >>> print(str(ticker.dt()))
```
6.2 Stock argument

6.3 Currencies arguments

6.4 Invalid

6.5 Class Ticker

This is description of abstract class.

```python
class ccs.abstract.Ticker (raw, symbol=None)
    Ticker like hell
    low()
        Low like hell :return:
    stock()
        Stock like hell :return:
```

6.6 Mapping
CHAPTER 7

Trades
Orderbook
9.1 Handling exceptions

Each description of function contains example of using. These examples are without exceptions treatment. Here is example for right using of `ticker()`.

```python
>>> import ccs
>>> response = ""
>>> try:
...     response = ccs.bitstamp.public.ticker("btcusd")
>>> except:
...     # handle exception
...     pass
```

9.2 How it works

These functions are only python wrappers around get requests. It means that here are not any extra controls parametres. Parametres which will give are strictly coding to GET request. It makes this functions flexible for changes. Typical situation it can be adding new symbol. On the other hand it mean you can write something like this:

```python
>>> import ccs
>>> ccs.bitstamp.public.transactions("btcusd", time="abcd")
```

It is clear that value of timestamp is not valid. However code doesn't invoke any exception. In fact it will return valid list of trades. But this behavior depends on Bitfinex server.

9.3 Response validation

If you would like validate response form server you can use prepared json schemas for validation. Here are examples for ticker method.

```python
>>> import ccs
>>> import json
>>> import jsonschema

>>> try:
...     response = ccs.bitstamp.public.ticker("btcusd")
...     schema = ccs.cfg.schema[ccs.constants.BITSTAMP]["ticker"]
```
Keys of schemas are absolutely same like name of functions. Schema of transaction request it can be read:

```python
>>> schema = ccs.cfg.schema[ccs.constants.BITSTAMP]["transactions"]
```

Validation is not built in function. The reason is bigger flexibility. It can happen that stock will change schema of json but API will stay same. For this and other situations validation is not implicit on this module level.

### 9.4 Congestion control

Last information is recommendation. These functions do not offer protection against congestion. It means you or your IP can be banned if you will send your requests very often. Send your requests in reasonable period. In fact here is not good reason ask every second on history of transactions, because stock with crypto currencies are not so liquid (without some Chinese stocks).
This implementation is build on Bitfinex REST API version 1.1. The official documentation is available here. It is highly recommended that the user should read General_informations before using.

10.1 Fundingbook

cms.bitfinex.public.fundingbook (currency, limit_bids=50, limit_asks=50)
The function provide information about full margin funding book.

Parameters

- currency (String) – This variable will contain values like btc, usd. For more information about symbols (currencies pairs) visit symbols() or symbolsDetails().
- limit_bids (Int) – It define maximum number of bids. This argument is optional. Default value is 50.
- limit_asks (Int) – It define maximum number of asks. This argument is optional. Default value is 50.

Returns

The function return payload of http response. It is string which particularly contains json with two lists of objects (dictionaries). Official description of object’s keys is in the table.

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bids</td>
<td>[array of funding bids]</td>
<td></td>
</tr>
<tr>
<td>rate</td>
<td>[rate in % per 365 days]</td>
<td></td>
</tr>
<tr>
<td>amount</td>
<td>[decimal]</td>
<td></td>
</tr>
<tr>
<td>period</td>
<td>[days]</td>
<td>Minimum period for the margin funding contract</td>
</tr>
<tr>
<td>timestamp</td>
<td>[time]</td>
<td></td>
</tr>
<tr>
<td>frr</td>
<td>[yes/no]</td>
<td>• Yes if the offer is at Flash Return Rate,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• No if the offer is at fixed rate</td>
</tr>
</tbody>
</table>

Asks
## Key

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>asks</td>
<td>[array of funding offers]</td>
<td></td>
</tr>
<tr>
<td>rate</td>
<td>[rate in % per 365 days]</td>
<td></td>
</tr>
<tr>
<td>amount</td>
<td>[decimal]</td>
<td></td>
</tr>
<tr>
<td>period</td>
<td>[days]</td>
<td>Maximum period for the funding contract</td>
</tr>
<tr>
<td>timestamp</td>
<td>[time]</td>
<td></td>
</tr>
<tr>
<td>frr</td>
<td>[yes/no]</td>
<td></td>
</tr>
</tbody>
</table>

- **Yes** if the offer is at Flash Return Rate,
- **No** if the offer is at fixed rate

### Return type

String

### Exception

It can raise any exception which can occur during using

- http.client.HTTPSConnection
- http.client.HTTPSConnection.request().

### Example

```python
>>> import ccs
>>> response = ccs.bitfinex.public.fundingbook("btc")
>>> print(response)
{
    "bids": [
        {
            "rate": "9.1287",
            "amount": "5000.0",
            "period": 30,
            "timestamp": "1444257541.0",
            "frr": "No"
        },
        ...
    ],
    "asks": [
        {
            "rate": "8.3695",
            "amount": "407.5",
            "period": 2,
            "timestamp": "1444260343.0",
            "frr": "No"
        },
        ...
    ]
}

>>> # Other examples of using
>>> ccs.bitfinex.public.fundingbook("btc", limit_asks=2)
>>> ccs.bitfinex.public.fundingbook("btc", limit_bids=2)
>>> ccs.bitfinex.public.fundingbook("btc", limit_asks=2, limit_bids=2)

>>> # Prepared validation schema
>>> schema = ccs.cfg.schema[ccs.constants.BITFINEX]["fundingbook"]
```
Note: This function use REST endpoint which is described on Bitfinex Fundingbook documentation.

Examples of GET request:

• https://api.bitfinex.com/v1/lendbook/btc
• https://api.bitfinex.com/v1/lendbook/btc?limit_asks=1
• https://api.bitfinex.com/v1/lendbook/btc?limit_bids=1
• https://api.bitfinex.com/v1/lendbook/btc?limit_bids=0
• https://api.bitfinex.com/v1/lendbook/btc?limit_asks=&limit_bids=1

10.2 Lends

ccs.bitfinex.public.lends(currency, timestamp=None, limit_lends=50)

The function provide a list of the most recent funding data for the given currency. It mean total amount provided and Flash Return Rate (in % by 365 days) over time.

Parameters

• currency (String) – This variable will contain values like btc, usd. For more information about symbols (currencies pairs) visit symbols() or symbolsDetails().

• timestamp (Number) – Setting this argument cause showing lends at or after the timestamp. This argument is optional.

• limit_lends (Int) – It define maximum number of lends. This argument is optional. Default value is 50.

Returns

The function return payload of http response. It is string which particularly contains json list of objects (dictionaries). Official description of object’s keys is in the table.

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rate</td>
<td>[decimal, % by 365 days]</td>
<td>Average rate of total funding received at fixed rates, ie past Flash Return Rate annualized</td>
</tr>
<tr>
<td>amount_lent</td>
<td>[decimal]</td>
<td>Total amount of open margin funding in the given currency</td>
</tr>
<tr>
<td>amount_used</td>
<td>[decimal]</td>
<td>Total amount of open margin funding used in a margin position in the given currency</td>
</tr>
<tr>
<td>timestamp</td>
<td>[time]</td>
<td></td>
</tr>
</tbody>
</table>
Return type  String
Exception  It can raise any exception which can occur during using
- http.client.HTTPSConnection
- http.client.HTTPSConnection.request().

Example

```python
>>> import ccs

>>> response = ccs.bitfinex.public.lends("btc")

>>> print(response)
[
    {
        "rate": "35.6443",
        "amount_lent": "15060.11291405",
        "amount_used": "14766.30959039",
        "timestamp": 1482168852
    }
    ...
]

>>> # Other examples of using
>>> ccs.bitfinex.public.lends("ltc", timestamp=1482168852)
>>> ccs.bitfinex.public.lends("etc", limit_lends=2)
>>> ccs.bitfinex.public.lends("btc", timestamp=1482168852, limit_lends=2)

>>> # Prepared validation schema
>>> schema = ccs.cfg.schema[ccs.constants.BITFINEX]["lends"]
```

Note: This function use REST endpoint which is described on Bitfinex Fundingbook documentation.

Examples of GET request:
- https://api.bitfinex.com/v1/lends/btc

10.3 Orderbook

**ccs.bitfinex.public.orderbook**(symbol, group=1, limit_bids=50, limit_asks=50)

This function provide actual lists of orders for sell and buy.

Parameters

- **symbol** (String) – Symbol is currency pair. For more information about symbols visit symbols() or symbolsDetails().
- **group** (Int) – If value is set on 1, orders are grouped by price in the orderbook.
  
  If value is set on 0, orders are not grouped and sorted individually.
  
  This argument is optional. Default value is 1.
• **limit_bids** (*Int*) – It define maximum number of bids. This argument is optional. Default value is 50.

• **limit_asks** (*Int*) – It define maximum number of asks. This argument is optional. Default value is 50.

**Returns**

The function return payload of http response. It is string which particularly contains json with two lists of objects (dictionaries). Official description of object’s keys is in the table.

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>price</td>
<td>[price]</td>
</tr>
<tr>
<td>amount</td>
<td>[decimal]</td>
</tr>
<tr>
<td>timestamp</td>
<td>[time]</td>
</tr>
</tbody>
</table>

**Return type** String

**Exception** It can raise any exception which can occur during using

• `http.client.HTTPSConnection`

• `http.client.HTTPSConnection.request()`.

**Example**

```python
>>> import ccs
>>> response = ccs.bitfinex.public.orderbook("btcusd")
>>> print(response)
{
    "bids":
        [
            {
                "price": "791.3",
                "amount": "11.86528138",
                "timestamp": "1482168501.0"
            },
            ...
        ],
    "asks":
        [
            {
                "price": "791.31",
                "amount": "11.76087989",
                "timestamp": "1482166207.0"
            },
            ...
        ]
}

>>> # Other examples of using
>>> ccs.bitfinex.public.orderbook("ltcusd", group=0)
>>> ccs.bitfinex.public.orderbook("ltcusd", limit_asks=2)
>>> ccs.bitfinex.public.orderbook("ltcusd", limit_bids=2)
>>> ccs.bitfinex.public.orderbook("ltcusd", limit_asks=2, limit_bids=2)
>>> ccs.bitfinex.public.orderbook("ltcusd", group=0, limit_asks=2, limit_bids=2)
>>> # Prepared validation schema
>>> schema = ccs.cfg.schema[ccs.constants.BITFINEX]["orderbook"]
```
Note: This function use REST endpoint which is described on Bitfinex Orderbook documentation.

Examples of GET request:

• https://api.bitfinex.com/v1/book/btcusd
• https://api.bitfinex.com/v1/book/btcusd?group=0
• https://api.bitfinex.com/v1/book/btcusd?limit_asks=2&limit_bids=2&group=0

10.4 Stats

ccs.bitfinex.public.stats(symbol)

The function provide statistics about symbol.

Parameters symbol (String) – Symbol is currency pair. For more information about symbols visit symbols() or symbolsDetails().

Returns

The function return payload of http response. It is string which contains json list of objects (dictionaries). One object (dictionary) represents statistic for period. Official description of object’s keys is in the table.

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>period</td>
<td>integer</td>
<td>Period covered in days</td>
</tr>
<tr>
<td>volume</td>
<td>price</td>
<td>Volume</td>
</tr>
</tbody>
</table>

Return type String

Exception It can raise any exception which can occur during using

• http.client.HTTPSConnection
• http.client.HTTPSConnection.request().

Example

```python
>>> import ccs
>>> response = ccs.bitfinex.public.stats("btcusd")
>>> print(response)
[
  {
    "period":1,
    "volume":"1814.47582303"
  },
  {
    "period":7,
    "volume":"28021.46283327"
  },
  {
    "period":30,
    "volume":"183014.34833507"
  }
```
Note: This function use REST endpoint which is described on Bitfinex Stats documentation.
Examples of GET request:

- https://api.bitfinex.com/v1/stats/btcusd

### 10.5 Symbols

**ccs.bitfinex.public.symbols()**

The function returns list of tradable currency pairs.

**Returns** The function returns payload of http response. It is string which contains json list of available symbols.

**Return type** String

**Exception** It can raise any exception which can occur during using

- http.client.HTTPSConnection
- http.client.HTTPSConnection.request()

**Example**

```python
>>> import ccs
>>> response = ccs.bitfinex.public.symbols()
>>> print(response)
[  "btcusd",
  "ltcusd",
  "ltcbtc",
  "ethusd",
  "ethbtc",
  "etcbtc",
  "etcnusd",
  "bfxusd",
  "bfxbtc",
  "rrtusd",
  "rrtbtc",
  "zecusd",
  "zebtc",
  "xmrusd",
  "xmrbtc"
]
```

Note: This function uses REST endpoint which is described on Bitfinex Symbols documentation.
Examples of GET request:

- https://api.bitfinex.com/v1/symbols

## 10.6 Symbols details

```python
ccs.bitfinex.public.symbolsDetails()
```

The function returns information about tradable pairs.

**Returns**

The function return payload of http response. It is string which particularly contains json list of objects (dictionaries). Official description of object’s keys is in the table.

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pair</td>
<td>string</td>
<td>The pair code</td>
</tr>
<tr>
<td>price_precision</td>
<td>integer</td>
<td>Maximum number of significant digits for price in this pair</td>
</tr>
<tr>
<td>initial_margin</td>
<td>decimal</td>
<td>Initial margin required to open a position in this pair</td>
</tr>
<tr>
<td>minimum_margin</td>
<td>decimal</td>
<td>Minimal margin to maintain (in %)</td>
</tr>
<tr>
<td>maximum_order_size</td>
<td>decimal</td>
<td>Maximum order size of the pair</td>
</tr>
<tr>
<td>expiration</td>
<td>string</td>
<td>Expiration date for limited contracts/pairs</td>
</tr>
</tbody>
</table>

**Return type** String

**Exception** It can raise any exception which can occur during using

- `http.client.HTTPSConnection`
- `http.client.HTTPSConnection.request()`.

**Example**

```python
>>> import ccs
>>> response = ccs.bitfinex.public.symbolsDetails()
>>> print(response)
[
  {
    "pair":"btcusd",
    "price_precision":5,
    "initial_margin":"30.0",
    "minimum_margin":"15.0",
    "maximum_order_size":"2000.0",
    "minimum_order_size":"0.01",
    "expiration":"NA"
  }
  ...
]

>>> # Prepared validation schema
>>> schema = ccs.cfg.schema[ccs.constants.BITFINEX]["symbolsDetails"]
```

**Note:** This function use REST endpoint which is described on Bitfinex Fundingbook documentation.
Examples of GET request:

- https://api.bitfinex.com/v1/symbols_details

10.7 Ticker

ccs.bitfinex.public.ticker(symbol)

This function provide tick data. This informations offer high level overview of the current states on the market. It is actual price, best bids and asks etc.

**Parameters** symbol (String) – Symbol is currency pair. For more information about symbols visit symbols() or symbolsDetails().

**Returns**

The function return payload of http response. It is string which contains json dictionary. Official description of keys is in the table.

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mid</td>
<td>[price]</td>
<td>(bid + ask) / 2</td>
</tr>
<tr>
<td>bid</td>
<td>[price]</td>
<td>Innermost bid</td>
</tr>
<tr>
<td>ask</td>
<td>[price]</td>
<td>Innermost ask</td>
</tr>
<tr>
<td>last_price</td>
<td>[price]</td>
<td>The price at which the last order executed</td>
</tr>
<tr>
<td>low</td>
<td>[price]</td>
<td>Lowest trade price of the last 24 hours</td>
</tr>
<tr>
<td>high</td>
<td>[price]</td>
<td>Highest trade price of the last 24 hours</td>
</tr>
<tr>
<td>volume</td>
<td>[price]</td>
<td>Trading volume of the last 24 hours</td>
</tr>
<tr>
<td>timestamp</td>
<td>[time]</td>
<td>The timestamp at which this information was valid</td>
</tr>
</tbody>
</table>

**Return type** String

**Exception** It can raise any exception which can occur during using

- http.client.HTTPSConnection
- http.client.HTTPSConnection.request().

**Example**

```python
>>> import ccs
>>> response = ccs.bitfinex.public.ticker("btcusd")
>>> print(response)
{"mid":"790.395","bid":"790.39","ask":"790.4","last_price":"790.28","low":"785.59","high":"792.27","volume":"1684.46613188","timestamp":"1482163796.189588406"}
```  
```python
>>> # Prepared validation schema
>>> schema = ccs.cfg.schema[ccs.constants.BITFINEX]["ticker"]
```

**Note:** This function use REST endpoint which is described on Bitfinex Ticker documentation.
Example of GET request:

• https://api.bitfinex.com/v1/pubticker/btcusd

10.8 Trades

ccs.bitfinex.public.trades(symbol, timestamp=None, limit_trades=None)

This function provide history of trades.

Parameters

• symbol (String) – Symbol is currency pair. For more information about symbols visit
  symbols() or symbolsDetails().

• timestamp (Number) – Setting this argument cause showing trades at or after the times-
  tamp. This argument is optional.

• limit_trades (Int) – It define maximum number of trades. This argument must be
  greater or equal to 1. This argument is optional. Default value is 50.

Returns

The function return payload of http response. It is string which contains json list of objects
(dictionaries). One object (dictionary) represents one trade. Official description of object’s keys
is in the table.

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tid</td>
<td>[integer]</td>
<td></td>
</tr>
<tr>
<td>timestamp</td>
<td>[time]</td>
<td></td>
</tr>
<tr>
<td>price</td>
<td>[price]</td>
<td></td>
</tr>
<tr>
<td>amount</td>
<td>[decimal]</td>
<td></td>
</tr>
<tr>
<td>exchange</td>
<td>[string]</td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>[string]</td>
<td>“sell” or “buy” (can be “” if undetermined)</td>
</tr>
</tbody>
</table>

Return type  String

Exception  It can raise any exception which can occur during using

• http.client.HTTPSConnection

• http.client.HTTPSConnection.request().

Example

```python
>>> import ccs
>>> response = ccs.bitfinex.public.trades("btcusd")
>>> print(response)
[  
  {  
    "timestamp":1482167987,
    "tid":25060454,
    "price":"790.94",
    "amount":"1.0",
    "exchange":"bitfinex",
    "type":"buy"
  },
  {  
    "timestamp":1482167919,
    "tid":25060449,
```
"price":"790.89",
"amount":"1.0",
"exchange":"bitfinex",
"type":"buy"
}
...

>>> # Other examples of using
>>> ccs.bitfinex.public.trades("ltcusd", timestamp=1482185015)
>>> ccs.bitfinex.public.trades("ltcusd", limit_trades=20)
>>> ccs.bitfinex.public.trades("ethusd", timestamp=1482185015, limit_trades=20)

# Prepared validation schema
>>> schema = ccs.cfg.schema[ccs.constants.BITFINEX]["trades"]

Note: This function use REST endpoint which is described on Bitfinex Trades documentation.

Examples of GET request:

- https://api.bitfinex.com/v1/trades/btcusd
- https://api.bitfinex.com/v1/trades/btcusd?timestamp=1
This implementation is build on Bitstamp REST API version 2. The official documentation is available here. It is highly recommended that the user should read General_informations before using.

### 11.1 EUR USD conversion rate

`ccs.bitstamp.public.eurUsdConversionRate()`

This function provide conversion rate between EUR and USD.

**Parameters** `symbol (String)` – Symbol is currency pair.

**Returns**

The function return payload of http response. It is string which contains json object. Official description of object’s keys is in the table.

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sell</td>
<td>price USD -&gt; EUR</td>
</tr>
<tr>
<td>buy</td>
<td>price EUR -&gt; USD</td>
</tr>
</tbody>
</table>

**Return type** String

**Exception** It can raise any exception which can occur during using

- `http.client.HTTPSConnection`
- `http.client.HTTPSConnection.request()`.

**Example**

```python
>>> import ccs
>>> response = ccs.bitstamp.public.eurUsdConversionRate()
>>> print(response)
{
    "sell": "1.0548",
    "buy": "1.0624"
}
>>> # Prepared validation schema
>>> schema = ccs.cfg.schema[ccs.constants.BITSTAMP]["eurUsdConversionRate"]
```

**Note:** This function use REST endpoint which is described on Bitstamp documentation.
Example of GET request:

- https://www.bitstamp.net/api/eur_usd/

## 11.2 Hourly ticker

**ccs.bitstamp.public.hourlyTicker**(symbol)

This function provides the same data as `ticker()`, but values are being calculated from within an hour.

**Parameters**

- **symbol** (String) – Symbol is currency pair.

**Returns**

The function returns the payload of the HTTP response. It is a string which contains a JSON dictionary. Official description of keys is in the table.

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>last</td>
<td>last BTC price</td>
</tr>
<tr>
<td>high</td>
<td>last hour price high</td>
</tr>
<tr>
<td>low</td>
<td>last hour price low</td>
</tr>
<tr>
<td>vwap</td>
<td>last hour volume weighted average price</td>
</tr>
<tr>
<td>volume</td>
<td>last hour volume</td>
</tr>
<tr>
<td>bid</td>
<td>highest buy order in actual hour</td>
</tr>
<tr>
<td>ask</td>
<td>lowest sell order in actual hour</td>
</tr>
<tr>
<td>timestamp</td>
<td>unix timestamp date and time</td>
</tr>
<tr>
<td>open</td>
<td>first price of the hour</td>
</tr>
</tbody>
</table>

**Return type**

String

**Exception**

It can raise any exception which can occur during using

- `http.client.HTTPSConnection`
- `http.client.HTTPSConnection.request()`.

**Example**

```python
>>> import ccs
>>> response = ccs.bitstamp.public.hourlyTicker("btcusd")
>>> print(response)
{
    "high": "906.00",
    "last": "890.02",
    "timestamp": "1483813890",
    "bid": "890.02",
    "vwap": "866.95",
    "volume": "23326.63588417",
    "low": "812.28",
    "ask": "890.84",
    "open": "904.95"
}
```

```python
>>> # Other examples of using
>>> ccs.bitstamp.public.hourlyTicker("btceur")
>>> ccs.bitstamp.public.hourlyTicker("eurusd")
```
Note: This function use REST endpoint which is described on Bitstamp documentation.

Example of GET request:
- https://www.bitstamp.net/api/v2/ticker_hour/btcusd/
- https://www.bitstamp.net/api/v2/ticker_hour/btceur/
- https://www.bitstamp.net/api/v2/ticker_hour/eurusd/

11.3 Orderbook

ccs.bitstamp.public.orderbook(symbol)

This function provide actual lists of orders for sell and buy.

Parameters symbol(String) – Symbol is currency pair.

Returns

The function return payload of http response. It is string which contains json object. Official description of object’s keys is in the table.

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>timestamp</td>
<td>unix timestamp</td>
</tr>
<tr>
<td>asks</td>
<td>list of sell orders</td>
</tr>
<tr>
<td>bids</td>
<td>list of buy orders</td>
</tr>
</tbody>
</table>

Each item in arrays for asks and bids describe one order. Official description of array position is in the table.

<table>
<thead>
<tr>
<th>Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>price</td>
</tr>
<tr>
<td>1</td>
<td>volume</td>
</tr>
</tbody>
</table>

Return type String

Exception It can raise any exception which can occur during using

- http.client.HTTPSConnection
- http.client.HTTPSConnection.request().

Example

```python
>>> import ccs
>>> response = ccs.bitstamp.public.orderbook("btcusd")
>>> print(response)
{ "timestamp": "1483817361",
  "bids":
  [ ["898.01", "7.55654329"],
    ["898.00", "2.24298440"],
    ...
  ],
```
"asks":
[
  ["898.51", "59.81171580"],
  ["898.52", "0.19552560"],
  ...
]
}

>>> # Prepared validation schema
>>> schema = ccs.cfg.schema[ccs.constants.BITSTAMP]["orderbook"]

Note: This function use REST endpoint which is described on Bitstamp documentation.

Example of GET request:

• https://www.bitstamp.net/api/v2/order_book/btcusd/
• https://www.bitstamp.net/api/v2/order_book/btceur/
• https://www.bitstamp.net/api/v2/order_book/eurusd/

11.4 Ticker

ccs.bitstamp.public.ticker(symbol)

This function provide tick data. This informations offer high level overview of the current states on the market.
It is actual price, best bids and asks etc.

Parameters symbol(String) – Symbol is currency pair.

Returns

The function return payload of http response. It is string which contains json dictionary. Official description of keys is in the table.

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>last</td>
<td>last BTC price</td>
</tr>
<tr>
<td>high</td>
<td>last 24 hours price high</td>
</tr>
<tr>
<td>low</td>
<td>last 24 hours price low</td>
</tr>
<tr>
<td>vwap</td>
<td>last 24 hours volume weighted average price</td>
</tr>
<tr>
<td>volume</td>
<td>last 24 hours volume</td>
</tr>
<tr>
<td>bid</td>
<td>highest buy order</td>
</tr>
<tr>
<td>ask</td>
<td>lowest sell order</td>
</tr>
<tr>
<td>timestamp</td>
<td>unix timestamp date and time</td>
</tr>
<tr>
<td>open</td>
<td>first price of the day</td>
</tr>
</tbody>
</table>

Return type String

Exception It can raise any exception which can occur during using

• http.client.HTTPSConnection
• http.client.HTTPSConnection.request().

Example
>>> import ccs
>>> response = ccs.bitstamp.public.ticker("btcusd")
>>> print(response)
{
    "high": "906.00",
    "last": "891.32",
    "timestamp": "1483813425",
    "bid": "889.33",
    "vwap": "867.24",
    "volume": "23430.28938458",
    "low": "812.28",
    "ask": "891.25",
    "open": "894.02"
}

# Other examples of using
>>> ccs.bitstamp.public.ticker("btceur")
>>> ccs.bitstamp.public.ticker("eurusd")

# Prepared validation schema
>>> schema = ccs.cfg.schema[ccs.constants.BITSTAMP]["ticker"]

Note: This function use REST endpoint which is described on Bitstamp documentation.
Example of GET request:

- https://www.bitstamp.net/api/v2/ticker/btcusd/
- https://www.bitstamp.net/api/v2/ticker/btceur/
- https://www.bitstamp.net/api/v2/ticker/eurusd/

11.5 Transactions

ccs.bitstamp.public.transactions(symbol, time=None)
This function provide history of trades.

Parameters

- symbol (String) – Symbol is currency pair.
- time (Integer) – It is Unix timestamp. Setting this argument cause showing trades at or after the time. This argument is optional.

Returns

The function return payload of http response. It is string which contains json array of objects. One object (dictionary) represents one trade. Official description of keys is in the table.

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>Unix timestamp date and time</td>
</tr>
<tr>
<td>tid</td>
<td>transaction ID</td>
</tr>
<tr>
<td>price</td>
<td>price</td>
</tr>
<tr>
<td>type</td>
<td>0 (buy) or 1 (sell)</td>
</tr>
</tbody>
</table>

Return type String

Exception It can raise any exception which can occur during using
http.client.HTTPSConnection
http.client.HTTPSConnection.request()

Example

```python
>>> import ccs
>>> response = ccs.bitstamp.public.transactions("btcusd")
>>> print(response)
[
  {
    "date": "1483816802",
    "tid": "12911918",
    "price": "898.01",
    "type": "1",
    "amount": "1.36000000"
  },
  {
    "date": "1483816801",
    "tid": "12911917",
    "price": "898.03",
    "type": "1",
    "amount": "0.15000000"
  },
  ...
]
>>> # Other examples of using
>>> ccs.bitstamp.public.transactions("btceur", time=1483813890)
>>> # Prepared validation schema
>>> schema = ccs.cfg.schema[ccs.constants.BITSTAMP]["transactions"]
```

**Note:** This function use REST endpoint which is described on Bitstamp documentation.

Example of GET request:

- https://www.bitstamp.net/api/v2/transactions/btcusd/
- https://www.bitstamp.net/api/v2/transactions/btcusd/?time=1483813890
12.1 Get currencies

`ccs.bittrex.public.getCurrencies()`  
This function provide informations about available currencies.

**Returns**

The function return payload of http response. It is string which contains json object. Official description of keys is in the table.

**TODO**

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Currency</td>
<td></td>
</tr>
<tr>
<td>CurrencyLong</td>
<td></td>
</tr>
<tr>
<td>MinConfirmation</td>
<td></td>
</tr>
<tr>
<td>TxFee</td>
<td></td>
</tr>
<tr>
<td>IsActive</td>
<td></td>
</tr>
<tr>
<td>CoinType</td>
<td></td>
</tr>
<tr>
<td>BaseAddress</td>
<td></td>
</tr>
<tr>
<td>Notice</td>
<td></td>
</tr>
</tbody>
</table>

**Return type** String

**Exception** It can raise any exception which can occur during using

- `http.client.HTTPSConnection`
- `http.client.HTTPSConnection.request()`

**Example**

```python
css >>> import ccss
>>> response = ccss.bittrex.public.getCurrencies()
>>> print(response)
{
  "success":true,
  "message":"",
  "result":
  {
    "Currency":"BTC",
    "CurrencyLong":"Bitcoin",
```
## 12.2 Get markets

ccs.bittrex.public.getMarkets()

This function provide informations about available marktets.

**Returns**

The function return payload of http response. It is string which contains json object. Official description of keys is in the table.

TODO

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MarketCurrency</td>
<td></td>
</tr>
<tr>
<td>BaseCurrency</td>
<td></td>
</tr>
<tr>
<td>MarketCurrencyLong</td>
<td></td>
</tr>
<tr>
<td>BaseCurrencyLong</td>
<td></td>
</tr>
<tr>
<td>MinTradeSize</td>
<td></td>
</tr>
<tr>
<td>MarketName</td>
<td></td>
</tr>
<tr>
<td>IsActive</td>
<td></td>
</tr>
<tr>
<td>Created</td>
<td></td>
</tr>
<tr>
<td>Notice</td>
<td></td>
</tr>
<tr>
<td>IsSponsored</td>
<td></td>
</tr>
<tr>
<td>LogoUrl</td>
<td></td>
</tr>
</tbody>
</table>
Return type String

Exception It can raise any exception which can occur during using
  • http.client.HTTPSConnection
  • http.client.HTTPSConnection.request().

Example

```python
>>> import ccs
>>> response = ccs.bittrex.public.getMarkets()
>>> print(response)
{
  "success":true,
  "message":"
  "result":
  [
    
    "MarketCurrency":"LTC",
    "BaseCurrency":"BTC",
    "MarketCurrencyLong":"Litecoin",
    "BaseCurrencyLong":"Bitcoin",
    "MinTradeSize":0.00000001,
    "MarketName":"BTC-LTC",
    "IsActive":true,
    "Created":"2014-02-13T00:00:00",
    "Notice":null,
    "IsSponsored":null,
    "LogoUrl":"https://i.imgur.com/R29q3dD.png"
  ],
  
  [ 
    
    "MarketCurrency":"DOGE",
    "BaseCurrency":"BTC",
    "MarketCurrencyLong":"Dogecoin",
    "BaseCurrencyLong":"Bitcoin",
    "MinTradeSize":0.00000001,
    "MarketName":"BTC-DOGE",
    "IsActive":true,
    "Created":"2014-02-13T00:00:00",
    "Notice":null,
    "IsSponsored":null,
    "LogoUrl":"https://i.imgur.com/e1RS4Hn.png"
  ],
  ...
  ]
}>>> # Prepared validation schema
>>> schema = ccs.cfg.schema[ccs.constants.BITTREX]["getMarkets"]
```

Note: This function use REST endpoint which is described on Bittrex documentation.

Example of GET request:
  •https://bittrex.com/api/v1.1/public/getmarkets
12.3 Get market history

ccs.bittrex.public.getMarketHistory(market, count=20)
This function provide history of trades.

Parameters

- **market** (String) – Market is currency pair. For more information about markets (symbols) visit getmarkets() orgetcurrencies().
- **count** (Integer) – It define maximum number of trades. This argument is optional. Default value is 20. Max is 50.

**Warning:** Count argument is mentioned in official documentation, but server absolutely ignore this value.

Returns

The function return payload of http response. It is string which contains json object. Official description of keys is in the table.

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td></td>
</tr>
<tr>
<td>TimeStamp</td>
<td></td>
</tr>
<tr>
<td>Quantity</td>
<td></td>
</tr>
<tr>
<td>Price</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
</tr>
<tr>
<td>FillType</td>
<td></td>
</tr>
<tr>
<td>OrderType</td>
<td></td>
</tr>
</tbody>
</table>

Return type  String

Exception  It can raise any exception which can occur during using

- http.client.HTTPSConnection
- http.client.HTTPSConnection.request().

Example

```python
>>> import ccs
>>> response = ccs.bittrex.public.getMarketHistory("btc-ltc")
>>> print(response)
{
  "success":true,
  "message":",
  "result":
  [
    {
      "Id":4126151,
      "Quantity":66597.09000000,
      "Price":0.00000025,
      "Total":0.01664927,
      "FillType":"FILL",
      "OrderType":"BUY"
    },
    {
```

Chapter 12. Bittrex
12.4 Get market summary

ccs.bittrex.public.getMarketSummary(market)

This function provide detailed data of give market. This informations offer high level overview of the current states on the market. It is actual price, best bids and asks etc.

Parameters market (String) – Market is currency pair. For more information about markets (symbols) visit getmarkets() or getcurrencies().

Returns

The function return payload of http response. It is string which contains json object. Official description of keys is in the table.

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MarketName</td>
<td>?</td>
</tr>
<tr>
<td>High</td>
<td>?</td>
</tr>
<tr>
<td>Low</td>
<td>?</td>
</tr>
<tr>
<td>Volume</td>
<td>?</td>
</tr>
<tr>
<td>Last</td>
<td>?</td>
</tr>
<tr>
<td>BaseVolume</td>
<td>?</td>
</tr>
<tr>
<td>TimeStamp</td>
<td>?</td>
</tr>
<tr>
<td>Bid</td>
<td>?</td>
</tr>
<tr>
<td>Ask</td>
<td>?</td>
</tr>
<tr>
<td>OpenBuyOrders</td>
<td>?</td>
</tr>
<tr>
<td>OpenSellOrders</td>
<td>?</td>
</tr>
<tr>
<td>PrevDay</td>
<td>?</td>
</tr>
<tr>
<td>Created</td>
<td>?</td>
</tr>
<tr>
<td>DisplayMarketName</td>
<td>?</td>
</tr>
</tbody>
</table>

Return type String
**Exception**  It can raise any exception which can occur during using

- http.client.HTTPSConnection
- http.client.HTTPSConnection.request().

**Example**

```python
>>> import ccs
>>> response = ccs.bittrex.public.getMarketSummary("btc-ltc")
>>> print(response)
{
  "success":true,
  "message":"
  "result":
  [
    {
      "MarketName":"BTC-LTC",
      "High":0.00454888,
      "Low":0.00423000,
      "Volume":1598.24416057,
      "Last":0.00436820,
      "BaseVolume":6.99532215,
      "TimeStamp":"2017-01-08T14:17:11.43",
      "Bid":0.00437737,
      "Ask":0.00440629,
      "OpenBuyOrders":170,
      "OpenSellOrders":859,
      "PrevDay":0.00448802,
      "Created":"2014-02-13T00:00:00"
    }
  ]
}
```

**Note:** This function use REST endpoint which is described on Bittrex documentation.

Example of GET request:


### 12.5 Get market summaries

**ccs.bittrex.public.getMarketSummaries()**

This function provide detailed data of all markets. This informations offer high level overview of the current states on the market. It is actual price, best bids and asks etc.

**Returns**

The function return payload of http response. It is string which contains json object. Official description of keys is in the table.
### Key
<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MarketName</td>
<td>?</td>
</tr>
<tr>
<td>High</td>
<td>?</td>
</tr>
<tr>
<td>Low</td>
<td>?</td>
</tr>
<tr>
<td>Volume</td>
<td>?</td>
</tr>
<tr>
<td>Last</td>
<td>?</td>
</tr>
<tr>
<td>BaseVolume</td>
<td>?</td>
</tr>
<tr>
<td>TimeStamp</td>
<td>?</td>
</tr>
<tr>
<td>Bid</td>
<td>?</td>
</tr>
<tr>
<td>Ask</td>
<td>?</td>
</tr>
<tr>
<td>OpenBuyOrders</td>
<td>?</td>
</tr>
<tr>
<td>OpenSellOrders</td>
<td>?</td>
</tr>
<tr>
<td>PrevDay</td>
<td>?</td>
</tr>
<tr>
<td>Created</td>
<td>?</td>
</tr>
<tr>
<td>DisplayMarketName</td>
<td>?</td>
</tr>
</tbody>
</table>

**Return type**  String

**Exception** It can raise any exception which can occur during using

- `http.client.HTTPSConnection`
- `http.client.HTTPSConnection.request()`.

**Example**

```python
>>> import ccs
>>> response = ccs.bittrex.public.getMarketSummaries()
>>> print(response)
{
    "success":true,
    "message":"
    "result":
    [
        {
            "MarketName":"BITCNY-BTC",
            "High":6000.00000001,
            "Low":6000.00000001,
            "Volume":0.00000020,
            "Last":6000.00000001,
            "BaseVolume":0.00120000,
            "TimeStamp":"2017-01-09T13:19:54.15",
            "Bid":6000.00000001,
            "Ask":59000.00000000,
            "OpenBuyOrders":15,
            "OpenSellOrders":14,
            "PrevDay":6000.00000001,
            "Created":"2015-12-11T06:31:40.653"
        },
        {
            "MarketName":"BTC-2GIVE",
            "High":0.00000049,
            "Low":0.00000038,
            "Volume":36247.97622366,
            "Last":0.000000038,
            "BaseVolume":0.01447996,
            "TimeStamp":"2017-01-09T13:29:46.937",
            "Bid":0.00000039,
            "Ask":0.00000040
        }
    ]
}
```
Note: This function use REST endpoint which is described on Bittrex documentation.
Example of GET request:


12.6 Get orderbook

ccs.bittrex.public.getOrderbook (market, depth=20, type='both')

This function provide actual lists of orders for sell and buy.

Parameters

- **market** (String) – Market is currency pair. For more information about markets (symbols) visit getmarkets() or getcurrencies().

- **depth** (Integer) – It define maximum number of asks / bids. This argument is optional. Default value is 20.

  Warning: Depth argument is mentioned in official documentation, but server absolutely ignore this value.

- **type** (String) – This argument identify type of orderbook. Available values are:
  
  - sell
  - buy
  - both

  This argument is optional. Default value is “both”.

Returns

The function return payload of http response. It is string which contains json object. Official description of keys is in the table.

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quantity</td>
<td></td>
</tr>
<tr>
<td>Rate</td>
<td></td>
</tr>
</tbody>
</table>

Return type String
Exception  It can raise any exception which can occur during using

- http.client.HTTPSConnection
- http.client.HTTPSConnection.request()

Example

```python
>>> import ccs
>>> response = ccs.bittrex.public.getOrderbook("btc-ltc")
>>> print(response)
{
    "success":true,
    "message":",
    "result":
    {
      "buy":
      [
        {"Quantity":0.12415465,"Rate":0.00437001},
        {"Quantity":3.58852516,"Rate":0.00435273},
        ...
      ],
      "sell":
      [
        {"Quantity":41.83912609,"Rate":0.00440900},
        {"Quantity":2.51315302,"Rate":0.00440904},
        ...
      ]
    }
}
```

```python
>>> # Other examples of using
>>> ccs.bittrex.public.getOrderbook("btc-ltc", depth=30)
>>> ccs.bittrex.public.getOrderbook("btc-ltc", type="sell")
>>> ccs.bittrex.public.getOrderbook("btc-ltc", type="buy")
>>> ccs.bittrex.public.getOrderbook("btc-ltc", depth=10, type="buy")
>>> # Prepared validation schema !! TYPE = BOTH
>>> schema = ccs.cfg.schema[ccs.constants.BITTREX]["getOrderbook"]
>>> # Prepared validation schema !! TYPE = BUY or TYPE=SELL
>>> schema = ccs.cfg.schema[ccs.constants.BITTREX]["getOrderbookBuySell"]
```

Note:  This function use REST endpoint which is described on Bittrex documentation.

Example of GET request:


### 12.7 Get ticker

ccs.bittrex.public.getTicker(market)
This function provide tick data. This informations offer high level overview.

**Parameters**  
*market* *(String)* – Market is currency pair. For more information about markets (symbols) visit `getmarkets()` or `getcurrencies()`.

**Returns**  
The function return payload of http response. It is string which contains json object. Official description of keys is in the table.

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bid</td>
<td></td>
</tr>
<tr>
<td>Ask</td>
<td></td>
</tr>
<tr>
<td>Last</td>
<td></td>
</tr>
</tbody>
</table>

**Return type**  
String

**Exception**  
It can raise any exception which can occur during using

- `http.client.HTTPSConnection`
- `http.client.HTTPSConnection.request()`

**Example**

```python
>>> import ccs
>>> response = ccs.bittrex.public.getTicker("btc-ltc")

>>> print(response)

{'success': True, 'message': '', 'result':
  {'Bid': 0.00436403,
   'Ask': 0.00441773,
   'Last': 0.00441777}
}
```

```python
>>> # Prepared validation schema
>>> schema = ccs.cfg.schema[ccs.constants.BITTREX]["getTicker"]
```

**Note:** This function use REST endpoint which is described on Bittrex documentation.  
Example of GET request:

This file implements functions for reading informations from Btcc-spot public REST endpoints.

```python
cs.btcc.public.orderbook (market='btccny', limit=None)
```

This function provide lists of orders for sell and buy.

**Parameters**

- **market** (*String*) – Market is currency pair.
- **limit** (*Integer*) – It define maximum number of asks and bids. This argument is optional.

**Returns**

The function return payload of http response. It is string which contains json object. Official description of keys is in the table.

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>asks</td>
<td>array</td>
<td></td>
</tr>
<tr>
<td>bids</td>
<td>array</td>
<td></td>
</tr>
<tr>
<td>date</td>
<td>number</td>
<td>last update timestamp</td>
</tr>
</tbody>
</table>

Each item in arrays for asks and bids describe one order. Official description of array position is in the table.

<table>
<thead>
<tr>
<th>Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>price</td>
</tr>
<tr>
<td>1</td>
<td>volume</td>
</tr>
</tbody>
</table>

**Return type** String

**Exception** It can raise any exception which can occur during using

- `http.client.HTTPSConnection`
- `http.client.HTTPSConnection.request()`.

**Example**

```python
>>> import ccs
>>> response = ccs.btcc.public.orderbook("btccny")
>>> print(response)
```
...,
"bids":
[
  [5721,0.6097],
  [5720.67,0.1],
  ...
],
"date":1484398991
}

Note: This function use REST endpoint which is described on Btcc-spot Orderbook documentation. Example of GET request:
•https://data.btcchina.com/data/orderbook?market=btccny
•https://data.btcchina.com/data/orderbook?market=btccny&limit=2

ccs.btcc.public.ticker (market='btccny')
This function provide detailed data of give market. This informations offer high level overview of the current states on the market. It is actual price, best bids and asks etc.

Parameters market (String) – Market is currency pair.

Returns
The function return payload of http response. It is string which contains json object. Official description of keys is in the table.

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>high</td>
<td>string</td>
<td>highest price in last 24h</td>
</tr>
<tr>
<td>low</td>
<td>string</td>
<td>lowest price in last 24h</td>
</tr>
<tr>
<td>buy</td>
<td>string</td>
<td>latest bid price</td>
</tr>
<tr>
<td>sell</td>
<td>string</td>
<td>latest ask price</td>
</tr>
<tr>
<td>last</td>
<td>string</td>
<td>last successful trade price</td>
</tr>
<tr>
<td>vol</td>
<td>string</td>
<td>total BTC volume in last 24h</td>
</tr>
<tr>
<td>date</td>
<td>number</td>
<td>last update timestamp</td>
</tr>
<tr>
<td>vwap</td>
<td>number</td>
<td>24 hour average filled price</td>
</tr>
<tr>
<td>prev_close</td>
<td>number</td>
<td>yesterday’s closed price</td>
</tr>
<tr>
<td>open</td>
<td>number</td>
<td>today’s opening price</td>
</tr>
</tbody>
</table>

Return type String

Exception It can raise any exception which can occur during using
• http.client.HTTPSConnection
• http.client.HTTPSConnection.request().

Example
```python
>>> import ccs

>>> response = ccs.btcc.public.ticker("btccny")

>>> response = ccs.btcc.public.ticker("ltccny")

>>> response = ccs.btcc.public.ticker("all")

>>> # Prepared validation schema

>>> schema = ccs.cfg.schema[ccs.constants.BTCC]["ticker"]
```

**Note:** This function use REST endpoint which is described on Btcc-spot Ticker documentation.

Example of GET request:

- `https://data.btccchina.com/data/ticker?market=all`
- `https://data.btccchina.com/data/ticker?market=btccny`
- `https://data.btccchina.com/data/ticker?market=ltccny`

**ccs.btcc.publictradeHistory** (*market* = `btccny`, *limit* = 100, *since* = None, *sincetype* = None)  
This function provide history of trades.

**Parameters**

- **market** (*String*) – Market is currency pair.
- **limit** (*Integer*) – It define maximum number of trades. This argument must be greater or equal to 1. This argument is optional. Default value is 100. Maximum is 5000.
- **since** (*Integer*) – Setting this argument cause showing trades at or after the timestamp or tid. This argument is optional.
- **sincetype** (*Integer*) – Available values for this argument are “id” or “time”. It specifies on which data the “since” parameter works. The default value is “id”.

**Returns**

The function return payload of http response. It is string which contains json object. Official description of keys is in the table.
<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>string</td>
<td>unix time in seconds since 1 January 1970</td>
</tr>
<tr>
<td>price</td>
<td>string</td>
<td>price for 1 BTC</td>
</tr>
<tr>
<td>amount</td>
<td>string</td>
<td>amount of BTC traded</td>
</tr>
<tr>
<td>tid</td>
<td>string</td>
<td>trade id</td>
</tr>
<tr>
<td>type</td>
<td>string</td>
<td>indicate 'buy' or 'sell' trade</td>
</tr>
</tbody>
</table>

**Return type**  String

**Exception**  It can raise any exception which can occur during using

- `http.client.HTTPSConnection`
- `http.client.HTTPSConnection.request()`.

**Example**

```python
>>> import ccs
>>> response = ccs.btcc.public.tradeHistory("btccny")
>>> print(response)
[
  {
    "date":"1484395859",
    "price":5679.94,
    "amount":0.064,
    "tid":"121327921",
    "type":"buy"
  },
  {
    "date":"1484395859",
    "price":5680.67,
    "amount":0.025,
    "tid":"121327922",
    "type":"buy"
  },
  ...
]
```

# Other examples of using
```python
>>> ccs.btcc.public.tradeHistory("ltccny", limit=10)
>>> ccs.btcc.public.tradeHistory("ltccny", since=7000)
>>> ccs.btcc.public.tradeHistory("ltccny", since=1484396000, sincetype="time")
>>> ccs.btcc.public.tradeHistory("ltccny", 10, 1484396000, "time")
```

# Prepared validation schema
```python
>>> schema = ccs.cfg.schema[ccs.constants.BTCC]["tradeHistory"]
```

**Note:**  This function use REST endpoint which is described on Btcc-spot Trade History documentation.

**Example of GET request:**

- `https://data.btcchina.com/data/historydata?market=btccny`
- `https://data.btcchina.com/data/historydata?market=ltccny`
- `https://data.btcchina.com/data/historydata?since=5000`
- `https://data.btcchina.com/data/historydata?since=5000&limit=10`
ccs.btcc.public.trades (market='btccny')

This function provide list of trades processed within the last 24h, but maximal number of trades returned is 10000.

**Parameters**  
market (String) – Market is currency pair.

**Returns**  
The function return payload of http response. It is string which contains json object. Official description of keys is in the table.

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>string</td>
<td>unix time in seconds since 1 January 1970</td>
</tr>
<tr>
<td>price</td>
<td>string</td>
<td>price for 1 BTC</td>
</tr>
<tr>
<td>amount</td>
<td>string</td>
<td>amount of BTC traded</td>
</tr>
<tr>
<td>tid</td>
<td>string</td>
<td>trade id</td>
</tr>
</tbody>
</table>

**Return type**  
String

**Exception**  
It can raise any exception which can occur during using

- http.client.HTTPSConnection
- http.client.HTTPSConnection.request().

**Example**

```python
>>> import ccs
>>> response = ccs.btcc.public.trades("btccny")
>>> print(response)
[{
   "date":"1484372797",
   "price":5615.41,
   "amount":0.029,
   "tid":"121266656"
},
{
   "date":"1484372797",
   "price":5615.53,
   "amount":0.371,
   "tid":"121266657"
},
...
]
```

```python
>>> # Other examples of using
>>> ccs.btcc.public.trades("ltccny")
>>> # Prepared validation schema
>>> schema = ccs.cfg.schema[ccs.constants.BTCC]["trades"]
```

**Note:**  
This function use REST endpoint which is described on Btcc-spot Trades documentation.

Example of GET request:

•https://data.btcchina.com/data/trades?market=ltccny
This file implements functions for reading informations from Btcc-spot public REST endpoints.

```python
ccs.btccpro.public.orderbook(symbol='XBTCNY', limit=None)
```

This function provide lists of orders for sell and buy.

**Parameters**

- `symbol (String)` – Symbol is currency pair.
- `limit (Integer)` – It define maximum number of asks and bids. This argument is optional.

**Returns**

The function return payload of http response. It is string which contains json object. Official description of keys is in the table.

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>asks</td>
<td>array</td>
<td></td>
</tr>
<tr>
<td>bids</td>
<td>array</td>
<td></td>
</tr>
<tr>
<td>date</td>
<td>number</td>
<td>last update timestamp</td>
</tr>
</tbody>
</table>

Each item in arrays for asks and bids describe one order. Official description of array position is in the table.

<table>
<thead>
<tr>
<th>Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>price</td>
</tr>
<tr>
<td>1</td>
<td>volume</td>
</tr>
</tbody>
</table>

**Return type** String

**Exception** It can raise any exception which can occur during using

- `http.client.HTTPSConnection`
- `http.client.HTTPSConnection.request()`.

**Example**

```python
>>> import ccs
>>> response = ccs.btccpro.public.orderbook("XBTCNY")
>>> print(response)
{
    "asks": [
        [5721.48, 0.8],
        [5721.4, 0.71],
        ...
    ],
    "bids": [
        [5721.4, 0.71],
        [5721.3, 0.8],
        ...
    ],
    "date": 1582509800
}
```
...]
    "bids":
    [
        [5721, 0.6097],
        [5720.67, 0.1],
        ...
    ],
    "date": 1484398991
}

>>> # Prepared validation schema
>>> schema = ccs.cfg.schema[ccs.constants.BTCCPRO]["orderbook"]

Note: This function use REST endpoint which is described on Btcc-pro orderbook documentation.

Example of GET request:

- https://pro-data.btcc.com/data/pro/orderbook?symbol=XBTCNY
- https://pro-data.btcc.com/data/pro/orderbook?limit=5&symbol=XBTCNY

ccs.btccpro.public.ticker(symbol='XBTCNY')
This function provide detailed data of give market. This informations offer high level overview of the current
states on the market. It is actual price, best bids and asks etc.

Parameters market (String) – Symbol is currency pair.

Returns

The function return payload of http response. It is string which contains json object. Official
description of keys is in the table.

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BidPrice</td>
<td>Double</td>
<td>bid price</td>
</tr>
<tr>
<td>AskPrice</td>
<td>Double</td>
<td>ask price</td>
</tr>
<tr>
<td>Open</td>
<td>Double</td>
<td>open price</td>
</tr>
<tr>
<td>High</td>
<td>Double</td>
<td>the highest trade price in 24 hours</td>
</tr>
<tr>
<td>Low</td>
<td>Double</td>
<td>the lowest trade price in 24 hours</td>
</tr>
<tr>
<td>Last</td>
<td>Double</td>
<td>last price</td>
</tr>
<tr>
<td>LastQuantity</td>
<td>Double</td>
<td>last quantity</td>
</tr>
<tr>
<td>PrevCls</td>
<td>Double</td>
<td>close Price</td>
</tr>
<tr>
<td>Timestamp</td>
<td>UTCTimestamp</td>
<td>timestamp</td>
</tr>
<tr>
<td>ExecutionLimitDown</td>
<td>Double</td>
<td>limit Down</td>
</tr>
<tr>
<td>ExecutionLimitUp</td>
<td>Double</td>
<td>limit Up</td>
</tr>
</tbody>
</table>

Return type String

Exception It can raise any exception which can occur during using

- http.client.HTTPSConnection
- http.client.HTTPSConnection.request().

Example

>>> import ccs
>>> response = ccs.btccpro.public.ticker("XBTCNY")
>>> print(response)
```json
{
  "ticker": {
    "BidPrice": 5553.5,
    "AskPrice": 5571,
    "Open": 5509.58,
    "High": 5610,
    "Low": 5450,
    "Last": 5571,
    "LastQuantity": 1,
    "PrevCls": 5581.9,
    "Volume": 2237,
    "Volume24H": 5200,
    "Timestamp": 1484478727152,
    "ExecutionLimitDown": 5408.8,
    "ExecutionLimitUp": 5743.4
  }
}

Note: This function use REST endpoint which is described on Btcc-pro Ticker documentation.

Example of GET request:
• https://pro-data.btcc.com/data/pro/ticker?symbol=XBTCNY

ccs.btccpro.public.tradeHistory (symbol='XBTCNY', limit=100, since=None, sincetype=None)
This function provide history of trades.

Parameters

• symbol (String) – Symbol is currency pair.

• limit (Integer) – It define maximum number of trades. This argument must be greater or equal to 1. This argument is optional. Default value is 100. Maximum is 5000.

• since (Integer) – Setting this argument cause showing trades at or after the timestamp or tid. This argument is optional.

• sincetype (Integer) – Available values for this argument are “id” or “time”. It specifies on which data the “since” parameter works. The default value is “id”.

Returns

The function return payload of http response. It is string which contains json object. Official description of keys is in the table.

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>String</td>
<td>trade id.</td>
</tr>
<tr>
<td>Timestamp</td>
<td>UTCTimestamp</td>
<td>Unix time in seconds since 1 January 1970.</td>
</tr>
<tr>
<td>Price</td>
<td>Double</td>
<td>trade price</td>
</tr>
<tr>
<td>Quantity</td>
<td>Double</td>
<td>trade quantity</td>
</tr>
<tr>
<td>Side</td>
<td>Char</td>
<td>sell or buy</td>
</tr>
</tbody>
</table>

Return type  String

Exception  It can raise any exception which can occur during using
- `http.client.HTTPSConnection`
- `http.client.HTTPSConnection.request()`

**Example**

```python
>>> import ccs
>>> response = ccs.btccpro.public.tradeHistory("XBTCNY")
>>> print(response)
[
  {
    "Id": 19,
    "Timestamp": 1456757388489,
    "Price": 2538,
    "Quantity": 2,
    "Side": "Sell"
  },
  ...
]
```

```python
>>> # Other examples of using
>>> ccs.btccpro.public.tradeHistory("ltccny", limit=10)
>>> ccs.btccpro.public.tradeHistory("ltccny", since=7000)
>>> ccs.btccpro.public.tradeHistory("ltccny", since=1484396000, sincetype="time")
>>> ccs.btccpro.public.tradeHistory("ltccny", 10, 1484396000, "time")
>>> # Prepared validation schema
>>> schema = ccs.cfg.schema[ccs.constants.BTCCPRO]["tradeHistory"]
```

**Note:** This function use REST endpoint which is described on Btcc-pro Trade History documentation.

Example of GET request:

- `https://pro-data.btcc.com/data/pro/historydata?symbol=XBTCNY`
- `https://pro-data.btcc.com/data/pro/historydata?limit=100&symbol=XBTCNY`
- `https://pro-data.btcc.com/data/pro/historydata?since=10&symbol=XBTCNY`
- `https://pro-data.btcc.com/data/pro/historydata?since=1484396000&symbol=XBTCNY`
- `https://pro-data.btcc.com/data/pro/historydata?since=1456757387645&limit=10&sincetype=time&symbol=XBTCNY`
This file implements functions for reading informations from Btcc-spot public REST endpoints.

\texttt{ccs.btccusd.public.orderbook(symbol='BTCUSD', limit=None)}

This function provide lists of orders for sell and buy.

\textbf{Parameters}

- \texttt{symbol (String)} – Symbol is currency pair.
- \texttt{limit (Integer)} – It define maximum number of asks and bids. This argument is optional.

\textbf{Returns}

The function return payload of http response. It is string which contains json object. Official description of keys is in the table.

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>asks</td>
<td>array</td>
<td></td>
</tr>
<tr>
<td>bids</td>
<td>array</td>
<td></td>
</tr>
<tr>
<td>date</td>
<td>number</td>
<td>last update timestamp</td>
</tr>
</tbody>
</table>

Each item in arrays for asks and bids describe one order. Official description of array position is in the table.

<table>
<thead>
<tr>
<th>Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>price</td>
</tr>
<tr>
<td>1</td>
<td>volume</td>
</tr>
</tbody>
</table>

\textbf{Return type} String

\textbf{Exception} It can raise any exception which can occur during using

- \texttt{http.client.HTTPSConnection}
- \texttt{http.client.HTTPSConnection.request()}.

\textbf{Example}

```python
>>> import ccs
>>> response = ccs.btccusd.public.orderbook("BTCUSD")
>>> print(response)
{'asks':
 ['[5721.48, 0.8],
  [5721.4, 0.71],
```
...}

Note: This function use REST endpoint which is described on ‘Btcc-usd orderbook documentation<https://www.btcc.com/apidocs/usd-spot-exchange-market-data-rest-api#order-book>’.

Example of GET request:
•https://spotusd-data.btcc.com/data/pro/orderbook?symbol=BTCUSD
•https://spotusd-data.btcc.com/data/pro/orderbook?symbol=BTCUSD&limit=5

ccs.btccusd.public.ticker(symbol=’BTCUSD’)
This function provide detailed data of give market. This informations offer high level overview of the current states on the market. It is actual price, best bids and asks etc.

Parameters market (String) – Symbol is currency pair.

Returns
The function return payload of http response. It is string which contains json object. Official description of keys is in the table.

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BidPrice</td>
<td>Double</td>
<td>bid price</td>
</tr>
<tr>
<td>AskPrice</td>
<td>Double</td>
<td>ask price</td>
</tr>
<tr>
<td>Open</td>
<td>Double</td>
<td>open price</td>
</tr>
<tr>
<td>High</td>
<td>Double</td>
<td>the highest trade price in 24 hours</td>
</tr>
<tr>
<td>Low</td>
<td>Double</td>
<td>the lowest trade price in 24 hours</td>
</tr>
<tr>
<td>Last</td>
<td>Double</td>
<td>last price</td>
</tr>
<tr>
<td>LastQuantity</td>
<td>Double</td>
<td>last quantity</td>
</tr>
<tr>
<td>PrevCls</td>
<td>Double</td>
<td>close Price</td>
</tr>
<tr>
<td>Timestamp</td>
<td>UTC/Timestamp</td>
<td>timestamp</td>
</tr>
<tr>
<td>ExecutionLimitDown</td>
<td>Double</td>
<td>limit Down</td>
</tr>
<tr>
<td>ExecutionLimitUp</td>
<td>Double</td>
<td>limit Up</td>
</tr>
</tbody>
</table>

Return type String

Exception It can raise any exception which can occur during using
• http.client.HTTPSConnection
• http.client.HTTPSConnection.request()

Example

>>> import ccs
g>>> response = ccs.btccusd.public.ticker("BTCUSD")
>>> print(response)
{
    "ticker":
    {
        "BidPrice":960.03,
        "AskPrice":1040,
        "Open":989.52,
        "High":1040,
        "Low":951.01,
        "Last":1040,
        "LastQuantity":0.138,
        "PrevCls":971.01,
        "Volume":4.8479,
        "Volume24H":5.2797,
        "Timestamp":1486037350348,
        "ExecutionLimitDown":841.87,
        "ExecutionLimitUp":1138.99
    }
}

# Prepared validation schema

```python
gaGetSymbol = ccs.cfg.schema[ccs.constants.BTCCUSD]["ticker"]
```

**Note:** This function use REST endpoint which is described on Btcc-usd Ticker documentation.

Example of GET request:

- `https://spotusd-data.btcc.com/data/pro/ticker?symbol=BTCUSD`

**ccs.btccusd.public.tradeHistory (symbol='BTCUSD', limit=100, since=None, sincetype=None)**

This function provide history of trades.

**Parameters**

- `symbol (String)` – Symbol is currency pair.
- `limit (Integer)` – It define maximum number of trades. This argument must be greater or equal to 1. This argument is optional. Default value is 100. Maximum is 5000.
- `since (Integer)` – Setting this argument cause showing trades at or after the timestamp or tid. This argument is optional.
- `sincetype (Integer)` – Available values for this argument are “id” or “time”. It specifies on which data the “since” parameter works. The default value is “id”.

**Returns**

The function return payload of http response. It is string which contains json object. Official description of keys is in the table.

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>String</td>
<td>trade id.</td>
</tr>
<tr>
<td>Timestamp</td>
<td>UTCTimestamp</td>
<td>Unix time in seconds since 1 January 1970.</td>
</tr>
<tr>
<td>Price</td>
<td>Double</td>
<td>trade price</td>
</tr>
<tr>
<td>Quantity</td>
<td>Double</td>
<td>trade quantity</td>
</tr>
<tr>
<td>Side</td>
<td>Char</td>
<td>sell or buy</td>
</tr>
</tbody>
</table>

**Return type** String

**Exception** It can raise any exception which can occur during using
• http.client.HTTPSConnection
• http.client.HTTPSConnection.request()

Example

```python
>>> import ccs
>>> response = ccs.btccusd.public.tradeHistory("BTCUSD")
>>> print(response)
[
      {
          "Id": 19,
          "Timestamp": 1456757388489,
          "Price": 2538,
          "Quantity": 2,
          "Side": "Sell"
      },
      ...
]
```

### Other examples of using

```python
>>> ccs.btccusd.public.tradeHistory("BTCUSD", limit=10)
>>> ccs.btccusd.public.tradeHistory("BTCUSD", since=7000)
>>> ccs.btccusd.public.tradeHistory("BTCUSD", since=1484396000, sincetype="time")
>>> ccs.btccusd.public.tradeHistory("BTCUSD", 10, 1484396000, "time")
```

### Prepared validation schema

```python
>>> schema = ccs.cfg.schema[ccs.constants.BTCCUSD]["tradeHistory"]
```

**Note:** This function uses REST endpoint which is described on Btcc-usd Trade History documentation.

Example of GET request:

- https://spotusd-data.btcc.com/data/pro/historydata?symbol=BTCUSD
- https://spotusd-data.btcc.com/data/pro/historydata?symbol=BTCUSD&limit=10
- https://spotusd-data.btcc.com/data/pro/historydata?symbol=BTCUSD&since=10
- https://spotusd-data.btcc.com/data/pro/historydata?symbol=BTCUSD&since=1484396000&limit=10&sincetype=time
16.1 Depth

ccs.btce.public.depth(pair, limit=150)
This function provide actual lists of orders for sell and buy.

Parameters

• pair (String) – For more information about symbols visit info().
• limit (Integer) – It define maximum number of trades. This argument must be greater or equal to 1. This argument is optional. Default value is 150. Maximum is 2000.

Returns

The function return payload of http response. It is string which contains json dictionary. Official description of keys is in the table.

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>asks</td>
<td>sell orders</td>
</tr>
<tr>
<td>bids</td>
<td>buy orders</td>
</tr>
</tbody>
</table>

Each item in arrays for asks and bids describe one order. Official description of array position is in the table.

<table>
<thead>
<tr>
<th>Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>price</td>
</tr>
<tr>
<td>1</td>
<td>volume</td>
</tr>
</tbody>
</table>

Return type  String

Exception  It can raise any exception which can occur during using

• http.client.HTTPSConnection
• http.client.HTTPSConnection.request().

Example

```python
>>> import ccs
>>> response = ccs.btce.public.depth("btc_usd")
>>> print(response)
```
[861.898, 0.8],
[861.899, 0.00002518],
...
"bids":
[
[860.1.01],
[859.203, 0.37],
...
]

Note: This function use REST endpoint which is described on Btce Depth documentation.

Example of GET request:
• https://btc-e.com/api/3/depth/btc_usd
• https://btc-e.com/api/3/depth/btc_usd?limit=2

16.2 Info

ccs.btce.public.info()

This function provide all the information about currently active pairs.

Returns

The function return payload of http response. It is string which contains json dictionary. Official description of keys is in the table.

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>decimal_places</td>
<td>number of decimals allowed during trading</td>
</tr>
<tr>
<td>min_price</td>
<td>minimum price allowed during trading</td>
</tr>
<tr>
<td>max_price</td>
<td>maximum price allowed during trading</td>
</tr>
<tr>
<td>min_amount</td>
<td>minimum sell or buy transaction size</td>
</tr>
<tr>
<td>hidden</td>
<td>whether the pair is hidden, 0 or 1</td>
</tr>
<tr>
<td>fee</td>
<td>commission for this pair</td>
</tr>
</tbody>
</table>

Return type String

Exception It can raise any exception which can occur during using
• http.client.HTTPSConnection
• http.client.HTTPSConnection.request().

Example
```python
>>> import ccs
>>> response = ccs.btce.public.info()
>>> print(response)
{
  "server_time":1483981601,
  "pairs":
  {
    "btc_usd":
    {
      "decimal_places":3,
      "min_price":0.1,
      "max_price":10000,
      "min_amount":0.01,
      "hidden":0,
      "fee":0.2
    },
    "btc_rur":
    {
      "decimal_places":5,
      "min_price":1,
      "max_price":1000000,
      "min_amount":0.01,
      "hidden":0,
      "fee":0.2
    },
    ...
  }
  ...
}
```

---

**Note:** This function use REST endpoint which is described on Btce Info documentation.

Example of GET request:

- `https://btc-e.com/api/3/info`

---

## 16.3 Ticker

ccs.btce.public.ticker(pair)

This function provide tick data. This informations offer high level overview of the current states on the market. It is actual price, best bids and asks etc.

**Parameters**

- **pair** (String) – For more information about symbols visit info().

**Returns**

The function return payload of http response. It is string which contains json dictionary. Official description of keys is in the table.
<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>high</td>
<td>maximum price</td>
</tr>
<tr>
<td>low</td>
<td>minimum price</td>
</tr>
<tr>
<td>avg</td>
<td>average price</td>
</tr>
<tr>
<td>vol</td>
<td>trade volume</td>
</tr>
<tr>
<td>vol_cur</td>
<td>trade volume in currency</td>
</tr>
<tr>
<td>last</td>
<td>the price of the last trade</td>
</tr>
<tr>
<td>buy</td>
<td>buy price</td>
</tr>
<tr>
<td>sell</td>
<td>sell price</td>
</tr>
<tr>
<td>updated</td>
<td>last update of cache</td>
</tr>
</tbody>
</table>

Return type  String

Exception  It can raise any exception which can occur during using

- http.client.HTTPSConnection
- http.client.HTTPSConnection.request().

Example

```python
>>> import ccs
>>> response = ccs.btce.public.ticker("btc_usd")
>>> print(response)
{
    "btc_usd":
    {
        "high":873,
        "low":840,
        "avg":856.5,
        "vol":4891718.21757,
        "vol_cur":5699.60085,
        "last":860.016,
        "buy":861.899,
        "sell":860.001,
        "updated":1483980795
    }
}
```

Note: This function use REST endpoint which is described on Btce Ticker documentation.

Example of GET request:

- https://btc-e.com/api/3/ticker/btc_usd

16.4 Trades

css.btce.public.trades(pair, limit=150)

This function provide history of trades.

Parameters

- **pair**(String) – For more information about symbols visit info().
• **limit** (*Integer*) – It defines the maximum number of trades. This argument must be greater or equal to 1. This argument is optional. Default value is 150. Maximum is 2000.

**Returns**

The function returns the payload of the HTTP response. It is a string containing a JSON dictionary. Official description of keys is in the table.

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>ask – sell, bid – buy.</td>
</tr>
<tr>
<td>price</td>
<td>buy price or sell price.</td>
</tr>
<tr>
<td>amount</td>
<td>the amount of asset bought/sold.</td>
</tr>
<tr>
<td>tid</td>
<td>trade ID.</td>
</tr>
<tr>
<td>timestamp</td>
<td>Unix time of the trade.</td>
</tr>
</tbody>
</table>

**Return type** *String*

**Exception** It can raise any exception which can occur during using

- `http.client.HTTPSConnection`
- `http.client.HTTPSConnection.request()`.

**Example**

```python
gt> import ccs
gt> response = ccs.btce.public.trades("btc_usd")
gt> print(response)
{
    "btc_usd":
    [       
        {          
            "type": "ask",          
            "price": 862,          
            "amount": 0.01396916,  
            "tid": 91331563,       
            "timestamp": 1483980974
        },       
        {          
            "type": "bid",          
            "price": 862.619,       
            "amount": 0.159,        
            "tid": 91331549,        
            "timestamp": 1483980971
        },       
        ...       
    ]
}
```  

Note: This function uses a REST endpoint which is described on Btce Trades documentation.  
Example of GET request:  
- `https://btc-e.com/api/3/trades/btc_usd`

16.4. Trades
• https://btc-e.com/api/3/trades/btc_usd?limit=2
17.1 Depth

\[ \text{ccs.bter.public.depth}(\text{symbol}) \]

This function provides actual lists of orders for sell and buy.

**Parameters**

symbol (String) – Symbol is currency pair. For more information about symbols visit tradingPairs().

**Returns**

The function returns the payload of the http response. It is a string which contains a json object. The official description of keys is in the table.

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>asks</td>
<td></td>
</tr>
<tr>
<td>bids</td>
<td></td>
</tr>
</tbody>
</table>

Each item in the arrays for asks and bids describes one order. The official description of array position is in the table.

<table>
<thead>
<tr>
<th>Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>price</td>
</tr>
<tr>
<td>1</td>
<td>volume</td>
</tr>
</tbody>
</table>

**Return type** String

**Exception** It can raise any exception which can occur during using

- http.client.HTTPSConnection
- http.client.HTTPSConnection.request().

**Example**

```python
>>> import ccs
>>> response = ccs.bter.public.depth("btc_cny")
>>> print(response)
{
    "result":"true",
    "asks":
    [
        [6390.57,1],
        [6389.63,0.8],
        ...
```
"bids":
[6300,0.501],
[6299.88,0.466],
...
}
}  

>>> # Prepared validation schema
>>> schema = ccs.cfg.schema[ccs.constants.BTER]["depth"]

Note: This function use REST endpoint which is described on Bter Depth documentation.
Example of GET request:

•http://data.bter.com/api/1/depth/btc_cny

17.2 Market info

ccs.bter.public.marketInfo()
This function provide informations about markets. Its are:
•market’s fee,
•minimum order total amount
•and price decimal places.

Returns

The function return payload of http response. It is string which contains json object. Official
description of keys is in the table.

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>decimal_places</td>
<td></td>
</tr>
<tr>
<td>min_amount</td>
<td></td>
</tr>
<tr>
<td>fee</td>
<td></td>
</tr>
</tbody>
</table>

Return type String

Exception It can raise any exception which can occur during using
• http.client.HTTPSConnection
• http.client.HTTPSConnection.request().

Example

```python
>>> import ccs
>>> response = ccs.bter.public.marketInfo()
>>> print(response)
```
"btc_cny":
{
   "decimal_places":2,
   "min_amount":0.5,
   "fee":0.2
}
},
"ltc_cny":
{
   "decimal_places":2,
   "min_amount":0.5,
   "fee":0.2
}
...

Note: This function use REST endpoint which is described on Bter Market Info documentation.
Example of GET request:
•http://data.bter.com/api/1/marketinfo

17.3 Market details

ccs.bter.public.marketDetails()
This function provide market details. http://data.bter.com/api/1/marketlist

Returns
The function return payload of http response. It is string which contains json object. Official
description of keys is in the table.
<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>no</td>
<td></td>
</tr>
<tr>
<td>symbol</td>
<td></td>
</tr>
<tr>
<td>name</td>
<td></td>
</tr>
<tr>
<td>name_cn</td>
<td></td>
</tr>
<tr>
<td>pair</td>
<td></td>
</tr>
<tr>
<td>rate</td>
<td></td>
</tr>
<tr>
<td>vol_a</td>
<td></td>
</tr>
<tr>
<td>vol_b</td>
<td></td>
</tr>
<tr>
<td>curr_a</td>
<td></td>
</tr>
<tr>
<td>curr_b</td>
<td></td>
</tr>
<tr>
<td>curr_suffix</td>
<td></td>
</tr>
<tr>
<td>rate_percent</td>
<td></td>
</tr>
<tr>
<td>trend</td>
<td></td>
</tr>
<tr>
<td>supply</td>
<td></td>
</tr>
<tr>
<td>marketcap</td>
<td></td>
</tr>
<tr>
<td>plot</td>
<td></td>
</tr>
</tbody>
</table>

**Return type**  String

**Exception**  It can raise any exception which can occur during using

- `http.client.HTTPSConnection`
- `http.client.HTTPSConnection.request()`.

**Example**

```python
>>> import ccs
>>> response = ccs.bter.public.marketDetails()
>>> print(response)
{
    "result":"true",
    "data": [
        {
            "no":1,
            "symbol":"ETC",
            "name":"Ethereum Classic",
            "name_cn": "",
            "pair":"etc_cny",
            "rate":"10.07",
            "vol_a":97079.3,
            "vol_b":973604,
            "curr_a":"ETC",
            "curr_b":"CNY",
            "curr_suffix":"CNY",
            "rate_percent":"0.90",
            "trend":"down",
            "supply":87687300,
            "marketcap":"883,011,111",
            "plot":null
        },
        {
            "no":2,
            "symbol":"BTC",
            "name":"Bitcoin",
            "name_cn": "",
```
"pair":"btc_cny",
"rate":"6255.71",
"vol_a":113.4,
"vol_b":"707,601",
"curr_a":"BTC",
"curr_b":"CNY",
"curr_suffix":" CNY",
"rate_percent":"0.01",
"trend":"down",
"supply":5249920,
"marketcap":"32,841,977,043",
"plot":null
},
...

Note: This function use REST endpoint which is described on Bter Market details documentation.

Example of GET request:

- `http://data.bter.com/api/1/marketlist`

17.4 Ticker

`ccs.bter.public.ticker(symbol)`

This function provide tick data. This informations offer high level overview of the current states on the market. It is actual price, best bids and asks etc.

**Parameters** `symbol` *(String)* – Symbol is currency pair. For more information about symbols visit `tradingPairs()`.

**Returns**

The function return payload of http response. It is string which contains json dictionary. Official description of keys is in the table.

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td></td>
</tr>
<tr>
<td>last</td>
<td></td>
</tr>
<tr>
<td>high</td>
<td></td>
</tr>
<tr>
<td>low</td>
<td></td>
</tr>
<tr>
<td>avg</td>
<td></td>
</tr>
<tr>
<td>sell</td>
<td></td>
</tr>
<tr>
<td>buy</td>
<td></td>
</tr>
<tr>
<td>vol_btc</td>
<td></td>
</tr>
<tr>
<td>vol_cny</td>
<td></td>
</tr>
<tr>
<td>rate_change_percentage</td>
<td></td>
</tr>
</tbody>
</table>

**Return type** String
Exception  It can raise any exception which can occur during using

• `http.client.HTTPSConnection`
• `http.client.HTTPSConnection.request()`.

Example

```python
>>> import ccs
>>> response = ccs.bter.public.ticker("btc_cny")
>>> print(response)
{
    "result":"true",
    "last":6301.94,
    "high":6440,
    "low":6050,
    "avg":6250.44,
    "sell":6304.44,
    "buy":6302.95,
    "vol_btc":129.367,
    "vol_cny":808581.69,
    "rate_change_percentage":"-1.41"
}
```

```python
>>> # Prepared validation schema
>>> schema = ccs.cfg.schema[ccs.constants.BTER]["ticker"]
```

Note: This function use REST endpoint which is described on Bter Ticker documentation.

Example of GET request:

• `http://data.bter.com/api/1/ticker/btc_cny`

17.5 Tickers

ccs.bter.public.tickers()

This function provide tick data for all markets (pairs). This informations offer high level overview of the current states on the market. It is actual price, best bids and asks etc.

Parameters `symbol` (String) – Symbol is currency pair. For more information about symbols visit `tradingPairs()`.

Returns

The function return payload of http response. It is string which contains json dictionary. Official description of keys is in the table.
## 17.5. Tickers

### Key

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td></td>
</tr>
<tr>
<td>last</td>
<td></td>
</tr>
<tr>
<td>high</td>
<td></td>
</tr>
<tr>
<td>low</td>
<td></td>
</tr>
<tr>
<td>avg</td>
<td></td>
</tr>
<tr>
<td>sell</td>
<td></td>
</tr>
<tr>
<td>buy</td>
<td></td>
</tr>
<tr>
<td>vol_btc</td>
<td></td>
</tr>
<tr>
<td>vol_cny</td>
<td></td>
</tr>
<tr>
<td>rate_change_percentage</td>
<td></td>
</tr>
</tbody>
</table>

**Return type**  
**String**

**Exception**  
It can raise any exception which can occur during using

- `http.client.HTTPSConnection`
- `http.client.HTTPSConnection.request()`.

**Example**

```python
>>> import ccs
>>> response = ccs.bter.public.tickers()
>>> print(response)
{
    "btc_cny":
    {
        "result":"true",
        "last":6204,
        "high":6367.49,
        "low":6050,
        "avg":6239.67,
        "sell":6222.32,
        "buy":6221.38,
        "vol_btc":113.564,
        "vol_cny":708601.96,
        "rate_change_percentage":"0.64"
    },
    "ltc_cny":
    {
        "result":"true",
        "last":27.44,
        "high":27.88,
        "low":27.2,
        "avg":27.57,
        "sell":27.5,
        "buy":27.44,
        "vol_ltc":2365.112,
        "vol_cny":65205.68,
        "rate_change_percentage":"-0.11"
    }
}
```

```python
# Prepared validation schema
>>> schema = ccs.cfg.schema[ccs.constants.BTER]["tickers"]
```
17.6 Trading pairs

ccs.bter.public.tradingPairs()

This function provide list of available trading pairs (symbols).

Returns The function return payload of http response. It is string which contains json array. Each item is trading pair.

Return type String

Exception It can raise any exception which can occur during using

  • http.client.HTTPSConnection
  • http.client.HTTPSConnection.request().

Example

```python
>>> import ccs
>>> response = ccs.bter.public.tradingPairs()
>>> print(response)
["btc_cny", "ltc_cny", "blk_cny", "btcny_cny", "bqc_cny", "bqb_cny", ...

>>> # Prepared validation schema
>>> schema = ccs.cfg.schema[ccs.constants.BTER]["tradingPairs"]
```
Returns

The function return payload of http response. It is string which contains json object. Official description of keys is in the table.

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td></td>
</tr>
<tr>
<td>price</td>
<td></td>
</tr>
<tr>
<td>tid</td>
<td></td>
</tr>
<tr>
<td>type</td>
<td></td>
</tr>
</tbody>
</table>

Return type  String

Exception  It can raise any exception which can occur during using

• http.client.HTTPSConnection
• http.client.HTTPSConnection.request().

Example

```python
>>> import ccs
>>> response = ccs.bter.public.tradeHistory("btc_cny")
>>> print(response)
{
  "result":"true",
  "data": [
    {
      "date":"1483966022",
      "price":6345.01,
      "amount":0.003,
      "tid":"425038",
      "type":"sell"
    },
    {
      "date":"1483966076",
      "price":6347.02,
      "amount":0.003,
      "tid":"425039",
      "type":"buy"
    },
    ...
  ],
  "elapsed":"0.054ms"
}
```

Note:  This function use REST endpoint which is described on Bter History documentation. Example of GET request:

• http://data.bter.com/api/1/trade/btc_cny
18.1 Chart

ccs.cexio.public.chart(cur1, cur2, lastHours, maxRespArrSize)

This function allows building price change charts (daily, weekly, monthly) and showing historical point in any point of the chart.

param String cur1 It is base currency. For more information about available currencies visit currencyLimits().

param String cur2 It is quote currency. For more information about available currencies visit currencyLimits().

param Integer lastHours Past tense period till the current date in hours.

param Integer maxRespArrSize Maximal amount of price values in return.

Returns

The function return payload of http response. It is string which contains json object. Official description of keys is in the table.

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tmsp</td>
<td>UNIX timestamp</td>
</tr>
<tr>
<td>price</td>
<td>price value</td>
</tr>
</tbody>
</table>

Return type String

Exception It can raise any exception which can occur during using

- http.client.HTTPSConnection
- http.client.HTTPSConnection.request().

Example

```python
>>> import ccs
>>> response = ccs.cexio.public.chart("BTC", "USD", 24, 100)
>>> print(response)
[
  {
    "tmsp":1482246000,
    "price":"796.658"
  },
]```
Note: This function use REST endpoint which is described on Cexio Chart documentation.
Here is not example, because this request is executed by POST method.

18.2 Convert

ccs.cexio.public.convert(cur1, cur2, amnt)

This function converts any amount of the currency to any other currency by multiplying the amount
by the last price of the chosen pair according to the current exchange rate.

**param String cur1**  It is base currency. For more information about available currencies
visit currencyLimits().

**param String cur2**  It is quote currency. For more information about available currencies
visit currencyLimits().

**param Integer amnt**  Amount of convertible currency.

**Returns**

The function return payload of http response. It is string which contains json object. Official
description of keys is in the table.

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>amnt</td>
<td>amount in the target currency</td>
</tr>
</tbody>
</table>

**Return type**  String

**Exception**  It can raise any exception which can occur during using

- http.client.HTTPSConnection
- http.client.HTTPSConnection.request().

**Example**

```python
>>> import ccs
>>> response = ccs.cexio.public.convert("BTC", "USD", 2.5)
>>> print(response)
{
    "amnt":2060.5
}
```

```python
>>> # Prepared validation schema
>>> schema = ccs.cfg.schema[ccs.constants.CEXIO]["convert"]
```
18.3 Currency limits

ccs.cexio.public.currencyLimits()

This function provide limits for all pairs.

Returns
The function return payload of http response. It is string which contains json objects. Each object describe one pair. Official description of keys is in the table.

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>symbol1</td>
<td>?</td>
</tr>
<tr>
<td>symbol2</td>
<td>?</td>
</tr>
<tr>
<td>minLotSize</td>
<td>?</td>
</tr>
<tr>
<td>minLotSizeS2</td>
<td>?</td>
</tr>
<tr>
<td>maxLotSize</td>
<td>?</td>
</tr>
<tr>
<td>minPrice</td>
<td>?</td>
</tr>
<tr>
<td>maxPrice</td>
<td>?</td>
</tr>
</tbody>
</table>

Return type  String

Exception  It can raise any exception which can occur during using
- http.client.HTTPSConnection
- http.client.HTTPSConnection.request().

Example
```python
>>> import ccs
>>> response = ccs.cexio.public.currencyLimits()
>>> print(response)
"e":"currency_limits",
"ok":"ok",
"data":
{
  "pairs":
  [
    { "symbol1":"BTC",
      "symbol2":"USD",
      "minLotSize":0.01,
      "minLotSizeS2":2.5,
      "maxLotSize":30,
      "minPrice":"1",
      "maxPrice":"4096"
    },
    { "symbol1":"BTC",
      "symbol2":"EUR",
      "minLotSize":0.01,
```
Note: This function use REST endpoint which is described on Cexio Currency limits documentation. Example of GET request:

•https://cex.io/api/currency_limits

18.4 Historical 1m OHLCV chart

ccs.cexio.public.historical1mOHLCVChart (cur1, cur2, year, month, day)

This function provides dataset to build 1m OHLCV chart for a given date. Relies on historical data, is not working for current date.

param String cur1 It is base currency. For more information about available currencies visit currencyLimits().

param String cur2 It is quote currency. For more information about available currencies visit currencyLimits().

param Integer year Integer value which represent year.

param Integer month Integer value which represent month.

param Integer day Integer value which represent day.

Returns

The function return payload of http response. It is string which contains json object. Official description of keys is in the table.

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>Unix timestamp</td>
</tr>
<tr>
<td>data1m</td>
<td>1440 candle data sets</td>
</tr>
</tbody>
</table>

Each item in arrays describe one 1m candle. Official description of array position is in the table.

<table>
<thead>
<tr>
<th>Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Unix timestamp</td>
</tr>
<tr>
<td>1</td>
<td>open price</td>
</tr>
<tr>
<td>2</td>
<td>high price</td>
</tr>
<tr>
<td>3</td>
<td>low price</td>
</tr>
<tr>
<td>4</td>
<td>close price</td>
</tr>
<tr>
<td>5</td>
<td>volume</td>
</tr>
</tbody>
</table>
Return type  String

Exception  It can raise any exception which can occur during using
  • http.client.HTTPSConnection
  • http.client.HTTPSConnection.request().

Example

```python
>>> import ccs
>>> response = ccs.cexio.public.historical1mOHLCVChart("BTC", "USD", 2016, 2, 28)
>>> print(response)
{
    "time":20160228,
    "data1m":
    [1456617600,434.3867,434.3867,433.781,433.781,4.15450000],
    [1456617660,433.747,433.747,433.7306,433.7306,3.00010001],
    ...
}
>>> # Prepared validation schema
>>> schema = ccs.cfg.schema[ccs.constants.CEXIO]["historical1mOHLCVChart"]
```

Note: This function use REST endpoint which is described on Cexio Historical 1m OHLCV Chart documentation.

Example of GET request:

*https://cex.io/api/ohlcv/hd/20160228/BTC/USD*

18.5 Last price

ccs.cexio.public.lastPrice(∗cur1, cur2∗)

This function provide last price for given market.

Parameters

• cur1 (String) – It is base currency. For more information about available currencies visit currencyLimits().

• cur2 (String) – It is quote currency. For more information about available currencies visit currencyLimits().

Returns

The function return payload of http response. It is string which contains json object. Official description of keys is in the table.

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>curr1</td>
<td>the first currency code;</td>
</tr>
<tr>
<td>curr2</td>
<td>the second currency code;</td>
</tr>
<tr>
<td>lprice</td>
<td>last price of selling/buying the first currency relative to the second one</td>
</tr>
</tbody>
</table>

18.5. Last price
Return type  String

Exception  It can raise any exception which can occur during using

- http.client.HTTPSConnection
- http.client.HTTPSConnection.request().

Example

```python
>>> import ccs
>>> response = ccs.cexio.public.lastPrice("BTC", "USD")
>>> print(response)
{
    "lprice":937.545,
    "curr1":"BTC",
    "curr2":"USD"
}

# Prepared validation schema

>>> schema = ccs.cfg.schema[ccs.constants.CEXIO]["lastPrice"]
```

Note: This function use REST endpoint which is described on Cexio Last price documentation. Example of GET request:

- https://cex.io/api/last_price/BTC/USD

### 18.6 Last prices for given market

`ccs.cexio.public.lastPricesForGivenMarket(*args)`

This function provide last price for required markets.

**Parameters**  `args` *(Array)* – It is array of strings, which contain name of currencies. For more information about available currencies visit `currencyLimits()`.

**Returns**

The function return payload of http response. It is string which contains json objects. Each object describe one ticker for one pair. Official description of keys is in the table.

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>symbol1</td>
<td>the first currency code;</td>
</tr>
<tr>
<td>symbol2</td>
<td>the second currency code;</td>
</tr>
<tr>
<td>lprice</td>
<td>last price of selling/buying the first currency relative to the second one</td>
</tr>
</tbody>
</table>

Return type  String

Exception  It can raise any exception which can occur during using

- http.client.HTTPSConnection
- http.client.HTTPSConnection.request().

Example
Note: This function use REST endpoint which is described on Cexio Last price for given market documentation.

Example of GET request:

- https://cex.io/api/last_prices/BTC/USD/LTC

18.7 Orderbook

ccs.cexio.public.orderbook(cur1, cur2, depth=None)

This function provide actual lists of orders for sell and buy.

Parameters

- **cur1** (String) – It is base currency. For more information about available currencies visit currencyLimits().

- **cur2** (String) – It is quote currency. For more information about available currencies visit currencyLimits().

- **depth** (Integer) – It define maximum number of asks / bids. This argument is optional.

Returns

The function return payload of http response. It is string which contains json object on top level.

Official description of keys is in the table.
<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>timestamp</td>
<td>Unix timestamp</td>
</tr>
<tr>
<td>bids</td>
<td>lists of open orders, each order is represented as a list</td>
</tr>
<tr>
<td>asks</td>
<td>lists of open orders, each order is represented as a list</td>
</tr>
<tr>
<td>pair</td>
<td>pair name</td>
</tr>
<tr>
<td>id</td>
<td>incremental version id of order-book snapshot, may be used to check changes</td>
</tr>
<tr>
<td>sell_total</td>
<td>total available in symbol1 (cur1)</td>
</tr>
<tr>
<td>buy_total</td>
<td>total available in symbol2 (cur2)</td>
</tr>
</tbody>
</table>

Each item in arrays for asks and bids describe one order. Official description of array position is in the table.

<table>
<thead>
<tr>
<th>Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>price</td>
</tr>
<tr>
<td>1</td>
<td>volume</td>
</tr>
</tbody>
</table>

**Return type**  String

**Exception**  It can raise any exception which can occur during using

- `http.client.HTTPSConnection`
- `http.client.HTTPSConnection.request()`.

**Example**

```python
>>> import ccs
>>> response = ccs.cexio.public.orderbook("BTC", "USD")
>>> print(response)
{
    "timestamp":1483868324,
    "bids":
    [
        [938.0029,0.05900835],
        [938.0027,0.01000000],
        ...
    ],
    "asks":
    [
        [940.0000,0.13479788],
        [941.1730,1.88500000],
        ...
    ],
    "pair":"BTC:USD",
    "id":26236005,
    "sell_total":"1212.64829285",
    "buy_total":"1293393.16"
}
```

```python
>>> # Other examples of using
>>> ccs.cexio.public.orderbook("BTC", "USD", depth=1)
```

```python
>>> # Prepared validation schema
>>> schema = ccs.cfg.schema[ccs.constants.CEXIO]["orderbook"]
```

**Note:**  This function use REST endpoint which is described on Cexio Orderbook documentation.

Example of GET request:

18.8 Ticker

cexio.public.ticker(cur1, cur2)

This function provide tick data. This informations offer high level overview of the current states on
the market. It is actual price, best bids and asks etc.

Parameters

- **cur1** (**String**) – It is base currency. For more information about available currencies
  visit currencyLimits().

- **cur2** (**String**) – It is quote currency. For more information about available currencies
  visit currencyLimits().

Returns

The function return payload of http response. It is string which contains json dictionary. Official
description of keys is in the table.

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>timestamp</td>
<td>unix timestamp</td>
</tr>
<tr>
<td>low</td>
<td>last 24 hours price low</td>
</tr>
<tr>
<td>high</td>
<td>last 24 hours price high</td>
</tr>
<tr>
<td>last</td>
<td>last BTC price</td>
</tr>
<tr>
<td>volume</td>
<td>last 24 hours volume</td>
</tr>
<tr>
<td>volume30d</td>
<td>last 30 days volume</td>
</tr>
<tr>
<td>bid</td>
<td>highest buy order</td>
</tr>
<tr>
<td>ask</td>
<td>lowest sell order</td>
</tr>
</tbody>
</table>

Return type  **String**

Exception  It can raise any exception which can occur during using

- http.client.HTTPSConnection

- http.client.HTTPSConnection.request().

Example

```python
>>> import ccs
>>> response = ccs.cexio.public.ticker("BTC", "USD")
>>> print(response)
{
    "timestamp": "1483867160",
    "low": "833",
    "high": "946.767",
    "last": "937.0052",
    "volume": "633.06282323",
    "volume30d": "16484.96095494",
    "bid": "937.0051",
    "ask": "937.5979"
}
```

Note: This function use REST endpoint which is described on Cexio Ticker documentation.
Example of GET request:

- [https://cex.io/api/ticker/BTC/USD](https://cex.io/api/ticker/BTC/USD)

## 18.9 Tickers for all pairs by market

```python
ccs.cexio.public.tickersForAllPairsByMarket(*args)
```

This function provide tick information for required markets.

**Parameters**

- **args** (*Array*) – It is array of strings, which contain name of currencies. For more information about available currencies visit `currencyLimits()`.

**Returns**

The function return payload of http response. It is string which contains json objects. Each object describe one ticker for one pair. Official description of keys is in the table.

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>timestamp</td>
<td>unix timestamp</td>
</tr>
<tr>
<td>low</td>
<td>last 24 hours price low</td>
</tr>
<tr>
<td>high</td>
<td>last 24 hours price high</td>
</tr>
<tr>
<td>last</td>
<td>last BTC price</td>
</tr>
<tr>
<td>volume</td>
<td>last 24 hours volume</td>
</tr>
<tr>
<td>volume30d</td>
<td>last 30 days volume</td>
</tr>
<tr>
<td>bid</td>
<td>highest buy order</td>
</tr>
<tr>
<td>ask</td>
<td>lowest sell order</td>
</tr>
</tbody>
</table>

**Return type** String

**Exception** It can raise any exception which can occur during using

- `http.client.HTTPSConnection`
- `http.client.HTTPSConnection.request()`.

**Example**

```python
>>> import ccs
>>> response = ccs.cexio.public.tickersForAllPairsByMarket("USD", "EUR", "RUB", "BTC")
>>> print(response)
```

```
{
"e":"tickers",
"ok":true,
"data": [
{
"timestamp":1483871358,
"pair":"BTC:USD",
"low":842.618,
"high":946.767,
```

```
"last": "937.545",
"volume": "628.64061219",
"volume30d": "16462.50176059",
"bid": "937.5473",
"ask": "938.9099"
},
{
"timestamp": "1483871358",
"pair": "LTC:USD",
"low": "3.6637",
"high": "4.104499",
"last": "4.0129",
"volume": "299.11955482",
"volume30d": "12250.58086773",
"bid": "3.963",
"ask": "4.04699999"
},
...

>>> # Prepared validation schema
>>> schema = ccs.cfg.schema[ccs.constants.CEXIO]["tickersForAllPairsByMarket"]

**Note:** This function use REST endpoint which is described on Cexio Ticker for all pairs documentation.

Example of GET request:

- https://cex.io/api/tickers/USD/EUR/RUB/BTC

---

18.10 Trade history

ccs.cexio.public.

**tradeHistory** *(cur1, cur2, since=None)*

This function provide history of trades.

**Parameters**

- **cur1** *(String)* – It is base currency. For more information about available currencies visit `currencyLimits()`.

- **cur2** *(String)* – It is quote currency. For more information about available currencies visit `currencyLimits()`.

- **since** *(Integer)* – Value of this argument is tid. Setting this argument cause showing trades with equal or higher tid. This argument is optional.

**Returns**

The function return payload of http response. It is string which contains json array of objects. Each object describe one trade. Official description of keys is in the table.
<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tid</td>
<td>unique trade id</td>
</tr>
<tr>
<td>type</td>
<td>buy or sell</td>
</tr>
<tr>
<td>amount</td>
<td>trade amount</td>
</tr>
<tr>
<td>price</td>
<td>price</td>
</tr>
<tr>
<td>date</td>
<td>Unix timestamp</td>
</tr>
</tbody>
</table>

Return type  String

Exception  It can raise any exception which can occur during using

- http.client.HTTPSConnection
- http.client.HTTPSConnection.request().

Example

```python
>>> import ccs
>>> response = ccs.cexio.public贸易History("BTC", "USD")
>>> print(response)
[{
   "type":"buy",
   "date":"1483867726",
   "amount":"0.13000000",
   "price":"937.0051",
   "tid":"1979261"
},
{
   "type":"sell",
   "date":"1483867558",
   "amount":"0.06504816",
   "price":"935.8778",
   "tid":"1979260"
},
...
]
```

```python
>>> # Other examples of using
>>> ccs.cexio.public贸易History("BTC", "USD", since=1)
```

```python
>>> # Prepared validation schema
>>> schema = ccs.cfg.schema[ccs.constants.CEXIO]["tradeHistory"]
```

Note:  This function use REST endpoint which is described on Cexio Trades history documentation.

Example of GET request:

- https://cex.io/api/trade_history/BTC/USD/
- https://cex.io/api/trade_history/BTC/USD/?since=1
API call rate limit
Every user of our API has a “call counter” which starts at 0.
Ledger/trade history calls increase the counter by 2.
Place/cancel order calls do not affect the counter.
All other API calls increase the counter by 1.
The user’s counter is reduced every couple of seconds, and if the counter exceeds the user’s maximum API access is suspended for 15 minutes. Tier 2 users have a maximum of 15 and their count gets reduced by 1 every 3 seconds.

19.1 Asset info

`ccs.kraken.public.getAssetInfo(info=None, aclass=None, asset=None)`

This function provide

**Parameters**

- `info (String)` – This argument is optional. Possible value is only `info`. It means it is not necessary. This argument is optional.
- `aclass (String)` – It is asset class. More oficial information are missing. Try to look example for better imagination. This argument is optional.
- `asset (String)` – This information are not official. It is analogy of currency with prefix “X” and “Z” for base and quote currency (aclass). Here is possible input array. Comma delimited list of assets to get info on (default = all for given asset class) This argument is optional.

**Returns**

The function return payload of http response. It is string which contains json objects. Each object describe one pair. Official description of keys is in the table.

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>altname</td>
<td>alternate name</td>
</tr>
<tr>
<td>aclass</td>
<td>asset class</td>
</tr>
<tr>
<td>decimals</td>
<td>scaling decimal places for record keeping</td>
</tr>
<tr>
<td>display_decimals</td>
<td>scaling decimal places for output display</td>
</tr>
</tbody>
</table>

**Return type** String
Exception  It can raise any exception which can occur during using

- http.client.HTTPSConnection
- http.client.HTTPSConnection.request().

Example

```python
>>> import ccs
>>> response = ccs.kraken.public.getAssetInfo()
>>> print(response)
{
  "error": [],
  "result": {
    "KFEF": {
       "aclass":"currency",
       "alname":"FEE",
       "decimals":2,
       "display_decimals":2
    },
    "XDAO": {
       "aclass":"currency",
       "alname":"DAO",
       "decimals":10,
       "display_decimals":3
    },
    ...
  }
}
```

```python
>>> # Other examples of using
>>> ccs.kraken.public.getAssetInfo(aclass="currency")
>>> ccs.kraken.public.getAssetInfo(asset="XXBT")
>>> ccs.kraken.public.getAssetInfo(asset="XXBT,ZEUR")
>>> ccs.kraken.public.getAssetInfo(aclss="currency", asset="XXBT")
>>> # Prepared validation schema
>>> schema = ccs.cfg.schema[ccs.constants.KRAKEN]["getAssetInfo"]
```

Note: This function use REST endpoint which is described on Kraken documentation.

Example of GET request:

- https://api.kraken.com/0/public/Assets
19.2 Asset pairs

ccs.kraken.public.getTradableAssetPairs(info=None, pair=None)

This function provide detailed information about asset pairs.

Parameters

- **info** *(String)* – This argument is optional. Possible values are in table.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>info</td>
<td>all info (default)</td>
</tr>
<tr>
<td>leverage</td>
<td>leverage info</td>
</tr>
<tr>
<td>fees</td>
<td>fees schedule</td>
</tr>
<tr>
<td>margin</td>
<td>margin info</td>
</tr>
</tbody>
</table>

- **pair** *(String)* – It is currency pair. This argument is optional.

Returns

The function return payload of http response. It is string which contains json objects. Each object describe one pair. Official description of keys is in the table.

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>altname</td>
<td>alternate pair name</td>
</tr>
<tr>
<td>aclass_base</td>
<td>asset class of base component</td>
</tr>
<tr>
<td>base</td>
<td>asset id of base component</td>
</tr>
<tr>
<td>aclass_quote</td>
<td>asset class of quote component</td>
</tr>
<tr>
<td>quote</td>
<td>asset id of quote component</td>
</tr>
<tr>
<td>lot</td>
<td>volume lot size</td>
</tr>
<tr>
<td>pair_decimals</td>
<td>scaling decimal places for pair</td>
</tr>
<tr>
<td>lot_decimals</td>
<td>scaling decimal places for volume</td>
</tr>
<tr>
<td>lot_multiplier</td>
<td>amount to multiply lot volume by to get currency volume</td>
</tr>
<tr>
<td>leverage_buy</td>
<td>array of leverage amounts available when buying</td>
</tr>
<tr>
<td>leverage_sell</td>
<td>array of leverage amounts available when selling</td>
</tr>
<tr>
<td>fees</td>
<td>fee schedule array in [volume, percent fee] tuples</td>
</tr>
<tr>
<td>fees_maker</td>
<td>maker fee schedule array in [volume, percent fee] tuples (if on maker/taker</td>
</tr>
<tr>
<td>fee_volume_currency</td>
<td>volume discount currency</td>
</tr>
<tr>
<td>margin_call</td>
<td>margin call level</td>
</tr>
<tr>
<td>margin_stop</td>
<td>stop-out/liquidation margin level</td>
</tr>
</tbody>
</table>

Return type  String

Exception  It can raise any exception which can occur during using

- http.client.HTTPSConnection
- http.client.HTTPSConnection.request().

Example

```python
>>> import ccs
>>> response = ccs.kraken.public.getTradableAssetPairs()
>>> print(response)
{
   "error": [],
   "result":
   { "XETCXETH":
```
{  
  "altname":"ETCETH",
  "aclass_base":"currency",
  "base":"XETC",
  "aclass_quote":"currency",
  "quote":"XETH",
  "lot":"unit",
  "pair_decimals":8,
  "lot_decimals":8,
  "lot_multiplier":1,
  "leverage_buy": [2],
  "leverage_sell": [2],
  "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]
  ],
  "fee_volume_currency":"ZUSD",
  "margin_call": 80,
  "margin_stop": 40
}

>>> # Other examples of using
>>> ccs.kraken.public.getTradableAssetPairs(info="leverage")
>>> ccs.kraken.public.getTradableAssetPairs(info="fees")
>>> ccs.kraken.public.getTradableAssetPairs(info="margin")
>>> ccs.kraken.public.getTradableAssetPairs(pair="XXBTZEUR")
>>> ccs.kraken.public.getTradableAssetPairs(pair="XXBTZEUR", info="leverage")

>>> # Prepared validation schema
>>> schema = ccs.cfg.schema[ccs.constants.KRAKEN]["getTradableAssetPairs"]

Note: This function use REST endpoint which is described on Kraken documentation.

Example of GET request:
19.3 OHLC

ccs.kraken.public.getOHLCdata(pair, interval=None, since=None)

This function provide candlestick chart.

Parameters

- **pair**: (String) – It is currency pair. For more information about symbols visit getTradableAssetPairs().
- **interval**: (Integer) – It is time frame interval in minutes. Possible values are 1 (default), 5, 15, 30, 60, 240, 1440, 10080, 21600.
- **since**: (Integer) – Value of since argument is trade ID. Setting this argument cause showing OHLC chart at or after the ID. This argument is optional.

Returns

The function return payload of http response. It is string which contains json object with arrays. Each array describe one time interval (one candle). Official description of array position is in the table.

<table>
<thead>
<tr>
<th>Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>time</td>
</tr>
<tr>
<td>1</td>
<td>open</td>
</tr>
<tr>
<td>2</td>
<td>high</td>
</tr>
<tr>
<td>3</td>
<td>low</td>
</tr>
<tr>
<td>4</td>
<td>close</td>
</tr>
<tr>
<td>5</td>
<td>vwap</td>
</tr>
<tr>
<td>6</td>
<td>volume</td>
</tr>
<tr>
<td>7</td>
<td>count</td>
</tr>
</tbody>
</table>

Key *last* is ID of last trade in answer from server. Note that ID can be used as since when polling for data.

Note: the last entry in the OHLC array is for the current, not-yet-committed frame and will always be present, regardless of the value of *since*.

Return type  String

Exception  It can raise any exception which can occur during using

- http.client.HTTPSConnection

Example

```python
>>> import ccs
>>> response = ccs.kraken.public.getOHLCdata("XBTEUR")
>>> print(response)
{
    "error": [],
```
"result":
{
 "XXBTZEUR":
[
  
  1482645840, 
  "834.000", 
  "834.000", 
  "834.000", 
  "834.000", 
  "834.000", 
  "0.07543179", 
  3 
],

  1482645900, 
  "834.000", 
  "834.000", 
  "833.100", 
  "833.999", 
  "833.166", 
  "0.42388696", 
  5 
],
...
],
"last":1482688920
}

Note: This function use REST endpoint which is described on Kraken documentation.
Example of GET request:
•https://api.kraken.com/0/public/OHLC?pair=XBTEUR
•https://api.kraken.com/0/public/OHLC?pair=XBTEUR&since=1482689400
•https://api.kraken.com/0/public/OHLC?pair=XBTEUR&since=1482689400

19.4 Orderbook

ccs.kraken.public.getOrderBook (pair, count=None)

This function provide actual lists of orders for sell and buy.

Parameters

• pair (String) – It is currency pair. For more information about symbols visit getTradableAssetPairs().
• **count (Integer)** – It define maximum number of asks / bids. This argument is optional.

**Returns**

The function return payload of http response. It is string which contains json object with arrays. Each array describe one order. Official description of array position is in the table. It is same for asks and bids.

<table>
<thead>
<tr>
<th>Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>price</td>
</tr>
<tr>
<td>1</td>
<td>volume</td>
</tr>
<tr>
<td>2</td>
<td>timestamp</td>
</tr>
</tbody>
</table>

**Return type** String

**Exception** It can raise any exception which can occur during using

- `http.client.HTTPSConnection`
- `http.client.HTTPSConnection.request()`

**Example**

```python
>>> import ccs
>>> response = ccs.kraken.public.getOrderBook("XBTEUR")
>>> print(response)
{
    "error": [],
    "result":
    {
        "XXBTZEUR":
        {
            "asks":
            [
                [
                    "863.24000","1.753",1482580426,
                    [
                        "863.61000","12.500",1482579746,
                        ...
                    ],
            "bids":
            [
                [
                    "862.00000","0.001",1482580604,
                    [
                        "861.48000","3.198",1482580657,
                    ...
                ]
            }
            }
        }
    }
}
```

**Note:** This function use REST endpoint which is described on Kraken documentation.

Example of GET request:

19.5 Ticker

ccs.kraken.public.getTickInformation(pair)

This function provides tick data. This information offers a high-level overview of the current states on the market. It includes actual price, best bids, and asks, etc.

Parameters:

- **pair** (String) – It is currency pair. For more information about symbols visit getTradableAssetPairs().

Returns

The function returns the payload of the HTTP response. It is a string containing a JSON dictionary. The official description of keys is in the table.

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>ask array(price, whole lot volume, lot volume)</td>
</tr>
<tr>
<td>b</td>
<td>bid array(price, whole lot volume, lot volume)</td>
</tr>
<tr>
<td>c</td>
<td>last trade closed array(price, lot volume)</td>
</tr>
<tr>
<td>v</td>
<td>volume array(today, last 24 hours)</td>
</tr>
<tr>
<td>p</td>
<td>volume weighted average price array(today, last 24 hours)</td>
</tr>
<tr>
<td>t</td>
<td>number of trades array(today, last 24 hours)</td>
</tr>
<tr>
<td>l</td>
<td>low array(today, last 24 hours)</td>
</tr>
<tr>
<td>h</td>
<td>high array(today, last 24 hours)</td>
</tr>
<tr>
<td>o</td>
<td>today’s opening price</td>
</tr>
</tbody>
</table>

Return type: String

Exception: It can raise any exception that can occur during using:
- http.client.HTTPSConnection
- http.client.HTTPSConnection.request().

Example:

```python
>>> import ccs
>>> response = ccs.kraken.public.getTickInformation("XBTEUR")
>>> print(response)
{
  "error": [],
  "result":
  {
    "XXBTZEUR":
    {
      "a": ["865.00000", "3", "3.000"],
      "b": ["863.00000", "5", "5.000"],
      "c": ["864.99900", "0.39297888"],
      "v": ["3028.35485167", "13443.20773038"],
      "p": ["871.88063", "867.96689"],
      "t": [3160, 13089],
      "l": ["857.00000", "833.94000"],
      "h": ["888.85900", "889.71200"],
      "o": "884.17900"
    }
  }
}
Note: This function use REST endpoint which is described on Kraken documentation.

Example of GET request:

• https://api.kraken.com/0/public/Ticker?pair=XBTEUR

19.6 Trades

ccs.kraken.public.getRecentTrades (pair, since=None)

This function provide history of trades.

Parameters

- **pair** (*String*) – It is currency pair. For more information about symbols visit getTradableAssetPairs().

- **since** (*Integer*) – Value of since argument is trade ID. Setting this argument cause showing trades at or after the ID. This argument is optional.

Returns

The function return payload of http response. It is string which contains json object with arrays. Each array describe one trade. Official description of array position is in the table.

<table>
<thead>
<tr>
<th>Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>price</td>
</tr>
<tr>
<td>1</td>
<td>volume</td>
</tr>
<tr>
<td>2</td>
<td>time</td>
</tr>
<tr>
<td>3</td>
<td>buy / sell</td>
</tr>
<tr>
<td>4</td>
<td>market / limit</td>
</tr>
<tr>
<td>5</td>
<td>miscellaneous</td>
</tr>
</tbody>
</table>

Key last is ID of last trade in answer from server. Note that ID can be used as since when polling for new trade data.

Return type String

Exception It can raise any exception which can occur during using

- http.client.HTTPSConnection

- http.client.HTTPSConnection.request().

Example

```python
>>> import ccs
>>> response = ccs.kraken.public.getRecentTrades("XBTEUR")
>>> print(response)
{"error":[]}
```
# Other examples of using
```python
ccs.kraken.public.getRecentTrades("XBTEUR", 1482576757925126325)
```  
# Prepared validation schema
```python
schema = ccs.cfg.schema[ccs.constants.KRAKEN]["getRecentTrades"]
```  
---

**Note:** This function use REST endpoint which is described on Kraken documentation.

Example of GET request:
- `https://api.kraken.com/0/public/Trades?pair=XBTEUR&since=1482576757925126325`

## 19.7 Server time

```python
ccs.kraken.public.getServerTime()
```  
This function provide server’s time.

**Returns** The function return payload of http response. It is string which contains json object. Time is provided in two formats. First is unix timestamp and second format is correspond standard rfc1123.

**Return type** String

**Exception** It can raise any exception which can occur during using
- `http.client.HTTPSConnection`
- `http.client.HTTPSConnection.request()`

**Example**
```python
>>> import ccs
>>> response = ccs.kraken.public.getServerTime()
>>> print(response)
{
  "error":[]
  "result":
  {
    "unixtime":1482674808,
    "rfc1123":"Sun, 25 Dec 16 14:06:48 +0000"
  }
}
```  
```python
>>> # Prepared validation schema
>>> schema = ccs.cfg.schema[ccs.constants.KRAKEN]["getServerTime"]
```
**Note:** This function use REST endpoint which is described on [Kraken documentation](https://www.kraken.com). Example of GET request:

```python
https://api.kraken.com/0/public/Time
```

### 19.8 Spread

**ccs.kraken.public.getRecentSpreadData**(pair, since=None)

This function provide spread data.

**Parameters**

- **pair** (String) – It is currency pair. For more information about symbols visit `getTradableAssetPairs()`.
- **since** (Integer) – Value of since argument is trade ID. Setting this argument cause showing spread data at or after the ID. This argument is optional.

**Returns**

The function return payload of http response. It is string which contains json object with arrays. Each array describe one time interval and its bid and ask. Official description of array position is in the table.

<table>
<thead>
<tr>
<th>Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>time</td>
</tr>
<tr>
<td>1</td>
<td>bid</td>
</tr>
<tr>
<td>2</td>
<td>ask</td>
</tr>
</tbody>
</table>

Key last is ID of last trade in answer from server. Note that ID can be used as since when polling for data.

**Return type** String

**Exception** It can raise any exception which can occur during using

- `http.client.HTTPSConnection`
- `http.client.HTTPSConnection.request()`.

**Example**

```python
>>> import ccs
>>> response = ccs.kraken.public.getRecentSpreadData("XBTEUR")
>>> print(response)
{
    "error": [],
    "result":
    {
        "XXBTZEUR":
        [
            [1482689922,"841.95000","843.00000"],
            [1482689932,"841.92900","843.00000"]
        ],
        "last":1482690474
    }
}
Note: This function use REST endpoint which is described on Kraken documentation.

Example of GET request:

•
20.1 Depth

ccs.okcoin.public.depth(symbol, size=None, merge=1)

This function provide actual lists of orders for sell and buy.

Parameters

- symbol (String) – Symbol is currency pair.
- size (Integer) – TODO value: must be between 1 - 200
- merge (Integer) – TODO value: 1, 0.1 (merge depth)

Returns

The function return payload of http response. It is string which contains json dictionary. Official description of keys is in the table.

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>asks</td>
<td>ask depth</td>
</tr>
<tr>
<td>bids</td>
<td>bid depth</td>
</tr>
</tbody>
</table>

Each item in arrays for asks and bids describe one order. Official description of array position is in the table.

<table>
<thead>
<tr>
<th>Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>price</td>
</tr>
<tr>
<td>1</td>
<td>volume</td>
</tr>
</tbody>
</table>

Return type String

Exception It can raise any exception which can occur during using

- http.client.HTTPSConnection
- http.client.HTTPSConnection.request()

Example

```python
>>> import ccs
>>> response = ccs.okcoin.com.public.depth("btc_usd")
>>> print(response)
{
    "asks":
    [
        [930.03,3],
        ...
```
[930,0.47],
...
],
"bids":
[
[889.24,0.284],
[889.02,0.336],
...
]
}

>>> # Other examples of using
>>> ccs.okcoin.cn.public.depth("btc_cny")
>>> ccs.okcoin.com.public.depth("btc_usd", size=2)
>>> ccs.okcoin.com.public.depth("btc_usd", merge=1)
>>> ccs.okcoin.com.public.depth("btc_usd", size=2, merge=1)

>>> # Prepared validation schema
>>> schema = ccs.cfg.schema[ccs.constants.OKCOINCN]["depth"]
>>> schema = ccs.cfg.schema[ccs.constants.OKCOINCOM]["depth"]

Note: This function use REST endpoint which is described on Okcoin documentation.

Example of GET request:
- https://www.okcoin.com/api/v1/depth.do?symbol=btc_usd&size=2&merge=1

20.2 Kline
ccs.okcoin.public.kline()
This function provide candlestick Data.

Returns
The function return payload of http response. It is string which contains json array. Each item in arrays describes one candle. Official description of array position is in the table.

<table>
<thead>
<tr>
<th>Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>timestamp</td>
</tr>
<tr>
<td>1</td>
<td>open</td>
</tr>
<tr>
<td>2</td>
<td>high</td>
</tr>
<tr>
<td>3</td>
<td>low</td>
</tr>
<tr>
<td>4</td>
<td>close</td>
</tr>
<tr>
<td>5</td>
<td>volume</td>
</tr>
</tbody>
</table>

Return type String

Warning: This function doesnt work.

Exception It can raise any exception which can occur during using
- http.client.HTTPSConnection
http.client.HTTPSConnection.request().

Example

```python
>>> import ccs
>>> response = ccs.okcoincom.public.kline()
>>> print(response)
[[1417478400000, 380.94, 387.7, 378.75, 384.61, 6857.31],
[1417564800000, 384.47, 387.13, 383.5, 387.13, 1062.04]]
```

Note: This function use REST endpoint which is described on Okcoin documentation.
Example of GET request:

```plaintext
•https://www.okcoin.com/api/v1/kline.do
```

### 20.3 Ticker

ccs.okcoin.public.ticker(symbol)

This function provide tick data. This informations offer high level overview of the current states on the market. It is actual price, best bids and asks etc.

**Parameters** symbol (String) – Symbol is currency pair.

**Returns**

The function return payload of http response. It is string which contains json dictionary. Official description of keys is in the table.
<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>server time for returned data</td>
</tr>
<tr>
<td>buy</td>
<td>best bid</td>
</tr>
<tr>
<td>high</td>
<td>highest price</td>
</tr>
<tr>
<td>last</td>
<td>latest price</td>
</tr>
<tr>
<td>low</td>
<td>lowest price</td>
</tr>
<tr>
<td>sell</td>
<td>best ask</td>
</tr>
<tr>
<td>vol</td>
<td>volume (in the last rolling 24 hours)</td>
</tr>
</tbody>
</table>

**Return type**  String

**Exception**  It can raise any exception which can occur during using

- `http.client.HTTPSConnection`
- `http.client.HTTPSConnection.request()`.

**Example**

```python
>>> import ccs
>>> response = ccs.okcoincom.public.ticker("btc_usd")
>>> print(response)
{
    "date":"1483982377",
    "ticker":
    {
        "buy":"893.01",
        "high":"912.0",
        "last":"894.0",
        "low":"862.91",
        "sell":"893.91",
        "vol":"2340.0015"
    }
}
```

```python
>>> # Other examples of using
>>> ccs.okcoincn.public.ticker("btc_cny")
>>> # Prepared validation schema
>>> schema = ccs.cfg.schema[ccs.constants.OKCOINCOM]["ticker"]
>>> schema = ccs.cfg.schema[ccs.constants.OKCOINCN]["ticker"]
```

**Note:** This function use REST endpoint which is described on Okcoin documentation.

Example of GET request:

- [https://www.okcoin.cn/api/v1/ticker.do?symbol=btc_cny](https://www.okcoin.cn/api/v1/ticker.do?symbol=btc_cny)
- [https://www.okcoin.cn/api/v1/ticker.do?symbol=ltc_cny](https://www.okcoin.cn/api/v1/ticker.do?symbol=ltc_cny)
20.4 Trades

ccs.okcoin.public.trades(symbol, since=1)

This function provide history of trades.

param String symbol  Symbol is currency pair.

param Integer since  Get recently 600 pieces of data starting from the given tid (optional).

return The function return payload of http response. It is string which contains json dictionary. Official description of keys is in the table.

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>transaction time</td>
</tr>
<tr>
<td>date_ms</td>
<td>transaction time in milliseconds</td>
</tr>
<tr>
<td>price</td>
<td>transaction price</td>
</tr>
<tr>
<td>amount</td>
<td>quantity in BTC (or LTC)</td>
</tr>
<tr>
<td>tid</td>
<td>transaction ID</td>
</tr>
<tr>
<td>type</td>
<td>buy/sell</td>
</tr>
</tbody>
</table>

rtype String

exception It can raise any exception which can occur during using

- http.client.HTTPSConnection
- http.client.HTTPSConnection.request().

Example

```python
>>> import ccs
>>> response = ccs.okcoincom.public.trades("btc_usd")
>>> print(response)
[[
  
  "amount":"0.099",
  "date":1483981229,
  "date_ms":1483981229000,
  "price":"887.22",
  "tid":208393434,
  "type":"sell"
],
[
  "amount":"0.705",
  "date":1483981229,
  "date_ms":1483981229000,
  "price":"887.01",
  "tid":208393436,
  "type":"sell"
],
...
]
```

>>> # Other examples of using
>>> ccs.okcoincn.public.trades("btc_cny")
>>> ccs.okcoincom.public.trades("ltc_usd", since=150)

>>> # Prepared validation schema
>>> schema = ccs.cfg.schema[ccs.constants.OKCOINCOM]["trades"]
>>> schema = ccs.cfg.schema[ccs.constants.OKCOINCN]["trades"]
Note: This function use REST endpoint which is described on Okcoin documentation.

Example of GET request:

- https://www.okcoin.com/api/v1/trades.do?symbol=ltc_usd
- https://www.okcoin.cn/api/v1/trades.do?symbol=btc_cny
- https://www.okcoin.cn/api/v1/trades.do?symbol=ltc_cny
21.1 Ticker

ccs.poloniex.public.returnTicker()

This function provide tick data. This informations offer high level overview of the current states on the market. It is actual price, best bids and asks etc.

Returns

The function return payload of http response for all markets (symbols). It is string which contains json dictionary of dictionary. Unofficial description of keys is in the table.

<table>
<thead>
<tr>
<th>Key</th>
<th>Unofficial description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>TODO</td>
</tr>
<tr>
<td>last</td>
<td>current price</td>
</tr>
<tr>
<td>lowestAsk</td>
<td>current lowest ask</td>
</tr>
<tr>
<td>highestBid</td>
<td>current highest bid</td>
</tr>
<tr>
<td>percentChange</td>
<td>percent change of price</td>
</tr>
<tr>
<td>baseVolume</td>
<td>volume of trades in base currency</td>
</tr>
<tr>
<td>quoteVolume</td>
<td>volume of trades in quote currency</td>
</tr>
<tr>
<td>isFrozen</td>
<td>TODO</td>
</tr>
<tr>
<td>high24hr</td>
<td>TODO</td>
</tr>
<tr>
<td>low24hr</td>
<td>TODO</td>
</tr>
</tbody>
</table>

Return type String

Exception It can raise any exception which can occur during using

* http.client.HTTPSConnection
* http.client.HTTPSConnection.request()

Example

```python
>>> import ccs
>>> response = ccs.poloniex.public.returnTicker()
>>> print(response)
{
    "BTC_1CR":
    {
        "id":1,
        "last":"0.00056825",
        "lowestAsk":"0.00056821",
```
"highestBid":"0.00051000",
"percentChange":"0.04266055",
"baseVolume":"0.75982797",
"quoteVolume":"1453.08528184",
"isFrozen":"0",
"high24hr":"0.00063000",
"low24hr":"0.00045303"
}

"BTC_BBR":
{
  "id":6,
  "last":"0.00008051",
  "lowestAsk":"0.00008065",
  "highestBid":"0.00008050",
  "percentChange":"-0.01372044",
  "baseVolume":"0.53542776",
  "quoteVolume":"6618.29854886",
  "isFrozen":"0",
  "high24hr":"0.00008222",
  "low24hr":"0.00008000"
}

Note: This function use REST endpoint which is described on Poloniex documentation.

Example of GET request:


21.2 Trade history

ccs.poloniex.public.returnTradeHistory(currencyPair, start=None, end=None)

This function provide history of trades.

Parameters

- **currencyPair** (String) – It is currency pair. For more information about each currency visit returnCurrencies(). For better imagination about pairs can be use returnTicker().

- **start** (Integer) – Start is UNIX timestamp. All trades which will return will have timestamp equal or higher. Here is one recommandation: test your window frame (start and end).

- **end** (Integer) – End is UNIX timestamp. All trades which will return will have timestamp equal or lower. Here is one recommandation: test your window frame (start and end).

Returns

The function return payload of http response. It is string which contains json array with object. Each object describe one trade. Unofficial description of array position is in the table.
Key | Description
---|---
globalTradeID | Unique ID across all markets on Poloniex
tradeID | Unique ID for this market (currency pair)
type | sell or buy
rate | equivalent for price
amount | amount
total | ?

Return type  String

Exception  It can raise any exception which can occur during using

- `http.client.HTTPSConnection`
- `http.client.HTTPSConnection.request()`

Example

```python
>>> import ccs
>>> response = ccs.poloniex.public.returnTradeHistory("BTC_LTC")
>>> print(response)
[
  {
    "globalTradeID":71118065,
    "tradeID":1094974,
    "date":"2016-12-26 10:25:11",
    "type":"buy",
    "rate":"895.70000000",
    "amount":"0.34670496",
    "total":"310.54363267"
  },
  {
    "globalTradeID":71118052,
    "tradeID":1094973,
    "date":"2016-12-26 10:25:04",
    "type":"buy",
    "rate":"895.70000000",
    "amount":"0.08561533",
    "total":"76.68565108"
  },
  ...
]
```

Note: This function use REST endpoint which is described on Poloniex documentation.

Example of GET request:

21.3 Orderbook

ccs.poloniex.public.returnOrderBook(currencyPair, depth=10)
This function provide actual lists of orders for sell and buy.

Parameters

- **currencyPair (String)** – It is currency pair. For more information about each currency visit returnCurrencies(). For better imagination about pairs can be use returnTicker().
- **depth (Integer)** – It define maximum number of asks / bids. Default vaule is 10.

Returns

The function return payload of http response. It is string which contains json object. Unofficial description of object’s keys is in the table.

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>asks</td>
<td>list of orders</td>
</tr>
<tr>
<td>bids</td>
<td>list of orders</td>
</tr>
<tr>
<td>isFrozen</td>
<td>?</td>
</tr>
<tr>
<td>seq</td>
<td>?</td>
</tr>
</tbody>
</table>

Each item in arrays for asks and bids describe one order. Unofficial description of array position is in the table.

<table>
<thead>
<tr>
<th>Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>price</td>
</tr>
<tr>
<td>1</td>
<td>volume</td>
</tr>
</tbody>
</table>

Return type String

Exception It can raise any exception which can occur during using

- http.client.HTTPSConnection
- http.client.HTTPSConnection.request().

Example

```python
g>>> import ccs
g>>> response = ccs.poloniex.public.returnOrderBook("BTC_LTC")
g>>> print(response)
{
    "asks": [
        ["0.00000689",4110.62513846],
        ["0.00000690",5557.36168574],
        ...
    ],
    "bids": [
        ["0.00000683",34.50893119],
        ["0.00000680",642.22946578],
        ...
    ],
    "isFrozen": "0",
    "seq": 23364099
}```
### Other examples of using

```python
ccs.poloniex.public.returnOrderBook("BTC_NXT", 30)
ccs.poloniex.public.returnOrderBook("all")
```

### Prepared validation schema

```python
schema = ccs.cfg.schema[ccs.constants.POLONIEX]["returnOrderBook"]
```

**Note:** This function use REST endpoint which is described on Poloniex documentation.

Example of GET request:


## 21.4 24h volume

**ccs.poloniex.public.return24hVolume()**

This function provide 24 hour volume for all markets and totals for primary currencies.

**Returns**

The function return payload of http response. It is string which contains json object. Unofficial

derivation of object’s keys is in the table.

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>base_&lt;quote&gt;</td>
<td>Json object contains 24 hours volumes for base and quote currency of pair.</td>
</tr>
<tr>
<td>total_base</td>
<td>Sum of volumes for base currency in last 24 hours.</td>
</tr>
</tbody>
</table>

*base* and <quote> represent currency like *BTC, LTC, ...*

**Return type**  String

**Exception** It can raise any exception which can occur during using

- http.client.HTTPSConnection
- http.client.HTTPSConnection.request()

**Example**

```python
>>> import ccs
>>> response = ccs.poloniex.public.return24hVolume()
>>> print(response)
```

```json
{"BTC_BBR": {
    "BTC": "8.21369390",
    "BBR": "75453.72075591"
},
"BTC_BCN": {
    "BTC": "1.90751575",
    "BCN": "34161303.95809131"
},
...,
```
"totalBTC":"26026.22129242",
"totalETH":"14592.70438383",
"totalUSDT":"5666182.79780848",
"totalXMR":"582.22698569",
"totalXUSD":"0.00000000"
}

>>> # Prepared validation schema
>>> schema = ccs.cfg.schema[ccs.constants.POLONIEX]["return24hVolume"]

Note: This function use REST endpoint which is described on Poloniex documentation.

Example of GET request:


21.5 Chart data

ccs.poloniex.public.returnChartData currencyPair, start, end, period=1800

This function provide candlestick chart data.

Parameters

- **currencyPair** (String) – It is currency pair. For more information about each currency visit returnCurrencies(). For better imagination about pairs can be use returnTicker().

- **start** (Integer) – Start is UNIX timestamp. All trades which will return will have timestamp equal or higher. Here is one recomendation: test your window frame (start and end).

- **end** (Integer) – End is UNIX timestamp. All trades which will return will have timestamp equal or lower. Here is one recomendation: test your window frame (start and end).

- **period** (Integer) – Time period of one candle. Valid period values are:
  - 300
  - 900
  - 1800
  - 7200
  - 14400
  - 86400

Values are in seconds. It coincides with 5 min, 15 min, 30 min, 2 hours, 4 hours and 24 hours.

Returns

The function return payload of http response. It is string which contains json array with object. Each object describe one trade. Unofficial escription of object’s keys is in the table.
<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>unix timestamp</td>
</tr>
<tr>
<td>high</td>
<td>candle attribute - higher price in period</td>
</tr>
<tr>
<td>low</td>
<td>candle attribute - lower price in period</td>
</tr>
<tr>
<td>open</td>
<td>candle attribute - opening price in period</td>
</tr>
<tr>
<td>close</td>
<td>candle attribute - closing price in period</td>
</tr>
<tr>
<td>volume</td>
<td>volume of base currency in period</td>
</tr>
<tr>
<td>quoteVolume</td>
<td>volume of quote currency in period</td>
</tr>
<tr>
<td>weightedAverage</td>
<td>weighted average in period</td>
</tr>
</tbody>
</table>

**Return type**  String

**Exception**  It can raise any exception which can occur during using
- `http.client.HTTPSConnection`
- `http.client.HTTPSConnection.request()`.

**Example**

```python
>>> import ccs
>>> response = ccs.poloniex.public.returnChartData("BTC_LTC", 1405699200, 9999999999, 300)
>>> print(response)
[
    {
        "date":1405699200,
        "high":0.01436175,
        "low":0.0140401,
        "open":0.01436175,
        "close":0.01436,
        "volume":0.39285884,
        "quoteVolume":27.6009686,
        "weightedAverage":0.01423351
    },
    {
        "date":1405713600,
        "high":0.0141799,
        "low":0.0141091,
        "open":0.01416,
        "close":0.0141799,
        "volume":0.17488903,
        "quoteVolume":12.37315145,
        "weightedAverage":0.01423455
    },
...
]

# Prepared validation schema
```n
`>>> schema = ccs.cfg.schema[ccs.constants.POLONIEX]["returnChartData"]`

**Note:** This function use REST endpoint which is described on Poloniex documentation.

Example of GET request:

21.6 Currencies

ccs.poloniex.public.returnCurrencies()

This function provide detail information about available currencies.

**return**  The function return payload of http response. It is string which contains json object of object. Each object describes one currency. Unofficial escription of object’s keys is in the table.

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>unique ID</td>
</tr>
<tr>
<td>name</td>
<td>full name of currency</td>
</tr>
<tr>
<td>txFee</td>
<td>fee</td>
</tr>
<tr>
<td>minConf</td>
<td>?</td>
</tr>
<tr>
<td>depositAddress</td>
<td>?</td>
</tr>
<tr>
<td>disabled</td>
<td>?</td>
</tr>
<tr>
<td>delisted</td>
<td>?</td>
</tr>
<tr>
<td>frozen</td>
<td>?</td>
</tr>
</tbody>
</table>

**rtype**  String

**exception**  It can raise any exception which can occur during using

- http.client.HTTPSConnection
- http.client.HTTPSConnection.request().

**Example**

```python
>>> import ccs
>>> response = ccs.poloniex.public.returnCurrencies()
>>> print(response)
{
   "1CR":
   {
      "id":1,
      "name":"1CRediT",
      "txFee":"0.01000000",
      "minConf":3,
      "depositAddress":null,
      "disabled":0,
      "delisted":1,
      "frozen":0
   },
   ...
   "BTC":
   {
      "id":28,
      "name":"Bitcoin",
      "txFee":"0.00010000",
      "minConf":1,
      "depositAddress":null,
      "disabled":0,
      "delisted":0,
      "frozen":0
   },
   ...
}
# Prepared validation schema

```python
>>> # Prepared validation schema
>>> schema = ccs.cfg.schema[ccs.constants.POLONIEX]["returnCurrencies"]
```

**Note:** This function use REST endpoint which is described on Poloniex documentation.

Example of GET request:


## 21.7 Loan orders

```python
ccs.poloniex.public.returnLoanOrders(currency)
```

This function provide list of loan offers and demands for a given currency.

**param String currency** For more information about available currencies visit `returnCurrencies()`.

**return** The function return payload of http response. It is string which contains json object of object. Each object describes one loan order. Unofficial escription of object’s keys is in the table.

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>offers</td>
<td></td>
</tr>
<tr>
<td>demands</td>
<td></td>
</tr>
</tbody>
</table>

### rtype String

**exception** It can raise any exception which can occur during using

- `http.client.HTTPSConnection`
- `http.client.HTTPSConnection.request()`

**Example**

```python
>>> import ccs
>>> response = ccs.poloniex.public.returnLoanOrders("BTC")
>>> print(response)
```

```json
{
  "offers": [
    {
      "rate": "0.00018500",
      "amount": "0.01487170",
      "rangeMin": 2,
      "rangeMax": 2
    }
  ]
}
```
Note: This function use REST endpoint which is described on Poloniex documentation. Example of GET request:

Indices and tables

- genindex
- modindex
- search
C
ccs.btcc.public, 49
ccs.btccpro.public, 55
ccs.btccusd.public, 59
S
stats() (in module ccs.bitfinex.public), 26
stock() (ccs.abstract.Ticker method), 14
symbols() (in module ccs.bitfinex.public), 27
symbolsDetails() (in module ccs.bitfinex.public), 28

T
Ticker (class in ccs.abstract), 14
ticker() (in module ccs.bitfinex.public), 29
ticker() (in module ccs.bitstamp.public), 36
ticker() (in module ccs.btcc.public), 50
ticker() (in module ccs.btccpro.public), 56
ticker() (in module ccs.btccusd.public), 60
ticker() (in module ccs.btce.public), 65
ticker() (in module ccs.bter.public), 73
ticker() (in module ccs.cexio.public), 87
ticker() (in module ccs.okcoin.public), 105
tickers() (in module ccs.bter.public), 74
tickersForAllPairsByMarket() (in module ccs.cexio.public), 88
tradeHistory() (in module ccs.btcc.public), 51
tradeHistory() (in module ccs.btccpro.public), 57
tradeHistory() (in module ccs.btccusd.public), 61
tradeHistory() (in module ccs.bter.public), 76
tradeHistory() (in module ccs.cexio.public), 89
trades() (in module ccs.bitfinex.public), 30
trades() (in module ccs.btcc.public), 53
trades() (in module ccs.btce.public), 66
trades() (in module ccs.okcoin.public), 107
tradingPairs() (in module ccs.bter.public), 76
transactions() (in module ccs.bitstamp.public), 37