Crypto Currencies Stocks Documentation

Release 0.1

Jan Seda

February 04, 2017

General:

CCS is Python package for communication with stocks which is traiding with crypto currencies.

CHAPTER 1

Overview

Crypto currencies stocks (ccs) is Python package for communication with stocks which is traiding 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. Unfication is associated with:

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

1.1 Supported stocks

Stock	Link
Bitfinex	https://www.bitfinex.com/
Bitstamp	https://www.bitstamp.net/
Bittrex	https://bittrex.com/
Btcc	https://www.btcc.com/
Btce	https://btc-e.com/
Bter	https://bter.com/
Cex.io	https://cex.io/
Kraken	https://www.kraken.com/
Okcoin.com	https://www.okcoin.com/
Okcoin.cn	https://www.okcoin.cn/
Poloniex	https://poloniex.com/

CHAPTER 2

Instalation

This package is part of pypi. Command for instalation is:

```
$ pip intall ccs
```

CHAPTER 3

Configuration

CHAPTER 4

Warnings

CHAPTER 5

Constants

CHAPTER 6

Ticker

6.1 Example of using

```
>>> 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.

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

CHAPTER 8

Orderbook

CHAPTER 9

General informations

9.1 Handeling exceptions

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

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

9.2 How it wokrs

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:

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

It is clear that value of timestamp is not valid. However code doesnt 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.

```
>>> import ccs
>>> import json
>>> import jsonschema
>>>
>>> try:
>>>     response = ccs.bitstamp.public.ticker("btcusd")
>>>     schema = ccs.cfg.schema[ccs.constants.BITSTAMP]["ticker"]
```

```
>>>     jsonschema.validate(json.loads(response), schema)
>>> except jsonschema.exceptions.ValidationError:
>>>     # handle validation exception
>>>     pass
>>> except:
>>>     # handle rest of exceptions (communication, ...)
>>>     pass
```

Keys of schemas are absolutely same like name of functions. Schema of transaction request it can be read:

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

Validation is not buil 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 recommandation. 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).

CHAPTER 10

Bifinex

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

ccs.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.

Bids

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

Asks

Key	Type	Description
asks	[array of funding offers]	
rate	[rate in % per 365 days]	
amount	[decimal]	
period	[days]	Maximum period for the funding contract
timestamp	[time]	
frr	[yes/no]	• **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

```
>>> 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.

Key	Type	Description
rate	[decimal, % by 365 days]	Average rate of total funding received at fixed rates, ie past Flash Return Rate annualized
amount_lent	[decimal]	Total amount of open margin funding in the given currency
amount_used	[decimal]	Total amount of open margin funding used in a margin position in the given currency
timestamp	[time]	

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.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
- https://api.bitfinex.com/v1/lends/btc?limit_lends=2
- https://api.bitfinex.com/v1/lends/btc?timestamp=1482000000
- https://api.bitfinex.com/v1/lends/btc?timestamp=1482000000&limit_lends=2

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.

Key	Type
price	[price]
amount	[decimal]
timestamp	[time]

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

Key	Type	Description
period	[integer]	Period covered in days
volume	[price]	Volume

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.bitfinex.public.stats("btcusd")
>>> print(response)
[
    {
        "period":1,
        "volume":"1814.47582303"
    },
    {
        "period":7,
        "volume":"28021.46283327"
    },
    {
        "period":30,
        "volume":"183014.34833507"
```

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

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 fucntion returns list of tradable currency pairs.

Returns The function return 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

```
    >>> import ccs
    >>> response = ccs.bitfinex.public.symbols()
    >>> print(response)
    [
        "btcusd",
        "ltcusd",
        "ltcbtc",
        "ethusd",
        "ethbtc",
        "etcbtc",
        "etcusd",
        "bfxusd",
        "bfxbtc",
        "rrtusd",
        "rrtbtc",
        "zecusd",
        "zecbtc",
        "xmrusd",
        "xmrbtc"
    ]
    >>>
    >>> # Prepared validation schema
    >>> schema = ccs.cfg.schema[ccs.constants.BITFINEX]["symbols"]
```

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

Examples of GET request:

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

10.6 Symbols details

ccs.bitfinex.public.**symbolsDetails**()

The function returns information about traidable 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.

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

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.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.

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

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.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"
}
>>>
>>> # 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 timestamp. 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.

Key	Type	Description
tid	[integer]	
timestamp	[time]	
price	[price]	
amount	[decimal]	
exchange	[string]	
type	[string]	"sell" or "buy" (can be "" if undetermined)

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.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?limit_trades=2
- https://api.bitfinex.com/v1/trades/btcusd?timestamp=1
- https://api.bitfinex.com/v1/trades/btcusd?timestamp=1&limit_trades=2

CHAPTER 11

Bitstamp

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.

Key	Description
sell	price USD -> EUR
buy	price EUR -> USDaa

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.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 provide same data as *ticker()*, but values are being calculated from within an hour.

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.

Key	Description
last	last BTC price
high	last hour price high
low	last hour price low
vwap	last hour volume weighted average price
volume	last hour volume
bid	highest buy order in actual hour
ask	lowest sell order in actual hour
timestamp	unix timestamp date and time
open	first price of the hour

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.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"
}
>>>
>>> # Other examples of using
>>> ccs.bitstamp.public.hourlyTicker("btceur")
>>> ccs.bitstamp.public.hourlyTicker("eurusd")
>>>
```

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

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.

Key	Description
timestamp	unix timestamp
asks	list of sell orders
bids	list of buy orders

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

Position	Description
0	price
1	volume

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.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.

Key	Description
last	last BTC price
high	last 24 hours price high
low	last 24 hours price low
vwap	last 24 hours volume weighted average price
volume	last 24 hours volume
bid	highest buy order
ask	lowest sell order
timestamp	unix timestamp date and time
open	first price of the day

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.

Key	Description
date	Unix timestamp date and time
tid	transaction ID
price	price
type	0 (buy) or 1 (sell)

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.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

CHAPTER 12

Bittrex

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

Key	Description
Currency	
CurrencyLong	
MinConfirmation	
TxFee	
IsActive	
CoinType	
BaseAddress	
Notice	

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.bittrex.public.getCurrencies()
>>> print(response)
{
    "success":true,
    "message":"",
    "result":
            [
                {
                    "Currency":"BTC",
                    "CurrencyLong":"Bitcoin",
```

```
                                "MinConfirmation":2,
                                "TxFee":0.00020000,
                                "IsActive":true,
                                "CoinType":"BITCOIN",
                                "BaseAddress":null,
                                "Notice":null
                            },
                            {
                                "Currency":"LTC",
                                "CurrencyLong":"Litecoin",
                                "MinConfirmation":6,
                                "TxFee":0.00200000,
                                "IsActive":true,
                                "CoinType":"BITCOIN",
                                "BaseAddress":null,
                                "Notice":null
                            },
                            ...
                        ]
    }
    >>>
    >>> # Prepared validation schema
    >>> schema = ccs.cfg.schema[ccs.constants.BITTREX]["getcurrencies"]
```

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

Example of GET request:

- https://bittrex.com/api/v1.1/public/getcurrencies

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

Key	Description
MarketCurrency	
BaseCurrency	
MarketCurrencyLong	
BaseCurrencyLong	
MinTradeSize	
MarketName	
IsActive	
Created	
Notice	
IsSponsored	
LogoUrl	

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.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()` or `getcurrencies()`.
- **count** (*Interger*) – 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.

TODO

Key	Description
Id	
TimeStamp	
Quantity	
Price	
Total	
FillType	
OrderType	

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.bittrex.public.getMarketHistory("btc-ltc")
>>> print(response)
{
    "success":true,
    "message":"",
    "result":
            [
                {
                    "Id":4126151,
                    "TimeStamp":"2017-01-09T12:32:18.377",
                    "Quantity":66597.09000000,
                    "Price":0.00000025,
                    "Total":0.01664927,
                    "FillType":"FILL",
                    "OrderType":"BUY"
                },
                {
```

```
                            "Id":4126114,
                            "TimeStamp":"2017-01-09T12:25:54.06",
                            "Quantity":23467.20827500,
                            "Price":0.00000024,
                            "Total":0.00563212,
                            "FillType":"PARTIAL_FILL",
                            "OrderType":"SELL"
                        }
                    ]
            }
            >>>
            >>> # Other examples of using
            >>> ccs.bittrex.public.getMarketHistory("btc-ltc", count=5)
            >>> # Prepared validation schema
            >>> schema = ccs.cfg.schema[ccs.constants.BITTREX]["getMarketHistory"]
```

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

Example of GET request:

- https://bittrex.com/api/v1.1/public/getmarkethistory?market=BTC-DOGE
- https://bittrex.com/api/v1.1/public/getmarkethistory?market=BTC-DOGE&count=2

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.

Key	Description
MarketName	?
High	?
Low	?
Volume	?
Last	?
BaseVolume	?
TimeStamp	?
Bid	?
Ask	?
OpenBuyOrders	?
OpenSellOrders	?
PrevDay	?
Created	?
DisplayMarketName	?

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.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"
                }
            ]
}
>>>
>>> # Prepared validation schema
>>> schema = ccs.cfg.schema[ccs.constants.BITTREX]["getMarketSummary"]
```

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

Example of GET request:

- https://bittrex.com/api/v1.1/public/getmarketsummary?market=btc-ltc

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	Description
MarketName	?
High	?
Low	?
Volume	?
Last	?
BaseVolume	?
TimeStamp	?
Bid	?
Ask	?
OpenBuyOrders	?
OpenSellOrders	?
PrevDay	?
Created	?
DisplayMarketName	?

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.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":0.00000038,
                    "BaseVolume":0.01447996,
                    "TimeStamp":"2017-01-09T13:29:46.937",
                    "Bid":0.00000039,
```

```
                            "Ask":0.00000043,
                            "OpenBuyOrders":52,
                            "OpenSellOrders":394,
                            "PrevDay":0.00000049,
                            "Created":"2016-05-16T06:44:15.287"
                        },
                        ...
                    ]
}
>>>
>>> # Prepared validation schema
>>> schema = ccs.cfg.schema[ccs.constants.BITTREX]["getMarketSummaries"]
```

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

Example of GET request:

- https://bittrex.com/api/v1.1/public/getmarketsummaries

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.

TODO

Key	Description
Quantity	
Rate	

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.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},
                        ...
                    ]
            }
}
>>>
>>> # 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:

- https://bittrex.com/api/v1.1/public/getorderbook?market=BTC-LTC&type=both
- https://bittrex.com/api/v1.1/public/getorderbook?market=BTC-LTC&type=buy
- https://bittrex.com/api/v1.1/public/getorderbook?market=BTC-LTC&type=sell
- https://bittrex.com/api/v1.1/public/getorderbook?market=BTC-LTC&type=both&depth=2

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.

Key	Description
Bid	
Ask	
Last	

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.bittrex.public.getTicker("btc-ltc")
>>> print(response)
{
    "success":true,
    "message":"",
    "result":
        {
            "Bid":0.00436403,
            "Ask":0.00441773,
            "Last":0.00441777
        }
}
>>>
>>> # 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:

- https://bittrex.com/api/v1.1/public/getticker?market=btc-ltc

CHAPTER 13

Btcc

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

ccs.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.

Key	Value	Description
asks	array	
bids	array	
date	number	last update timestamp

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

Position	Description
0	price
1	volume

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.btcc.public.orderbook("btccny")
>>> print(response)
{
    "asks":
            [
                [5721.48,0.8],
                [5721.4,0.71],
```

```
                        ...
                    ],
                "bids":
                    [
                        [5721,0.6097],
                        [5720.67,0.1],
                        ...
                    ],
                "date":1484398991
            }
            >>>
            >>> # 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 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.

Key	Value	Description
high	string	highest price in last 24h
low	string	lowest price in last 24h
buy	string	latest bid price
sell	string	latest ask price
last	string	last successful trade price
vol	string	total BTC volume in last 24h
date	number	last update timestamp
vwap	number	24 hour average filled price
prev_close	number	yesterday's closed price
open	number	today's opening price

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.btcc.public.ticker("btccny")
>>> print(response)
{
    "ticker":
            {
                "high":"5720.00",
                "low":"5325.01",
                "buy":"5646.33",
                "sell":"5647.18",
                "last":"5646.33",
                "vol":"886404.27650000",
                "date":1484389306,
                "vwap":"5654",
                "prev_close":"5625.78",
                "open":"5625.98"
            }
}
>>>
>>> # Other examples of using
>>> ccs.btcc.public.ticker("ltccny")
>>> 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.btcchina.com/data/ticker?market=all
- https://data.btcchina.com/data/ticker?market=btccny
- https://data.btcchina.com/data/ticker?market=ltccny

ccs.btcc.public.**tradeHistory**(*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.

Key	Value	Description
date	string	unix time in seconds since 1 January 1970
price	string	price for 1 BTC
amount	string	amount of BTC traded
tid	string	trade id
type	string	indicate 'buy' or 'sell' trade

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.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
>>> 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
>>> 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?marktet=btccny
- https://data.btcchina.com/data/historydata?marktet=ltccny
- https://data.btcchina.com/data/historydata?limit=10
- https://data.btcchina.com/data/historydata?since=5000
- https://data.btcchina.com/data/historydata?since=5000&limit=10

•https://data.btcchina.com/data/historydata?since=1406794449&limit=10&sincetype=time

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.

Key	Value	Description
date	string	unix time in seconds since 1 January 1970
price	string	price for 1 BTC
amount	string	amount of BTC traded
tid	string	trade id

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.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"
    },
    ...
]
>>>
>>> # 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=btccny

- https://data.btcchina.com/data/trades?market=ltccny

Btccpro

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

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.

Key	Value	Description
asks	array	
bids	array	
date	number	last update timestamp

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

Position	Description
0	price
1	volume

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.orderbook("XBTCNY")
>>> print(response)
{
    "asks":
            [
                [5721.48,0.8],
                [5721.4,0.71],
```

```
                        ...
                    ],
            "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.

Key	Value	Description
BidPrice	Double	bid price
AskPrice	Double	ask pric
Open	Double	open price
High	Double	the highest trade price in 24 hours
Low	Double	the lowest trade price in 24 hours
Last	Double	last price
LastQuantity	Double	last quantity
PrevCls	Double	close Price
Timestamp	UTCTimestamp	timestamp
ExecutionLimitDown	Double	limit Down
ExecutionLimitUp	Double	limit Up

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)
```

```
{
    "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
            }
}
>>>
>>> # Prepared validation schema
>>> schema = ccs.cfg.schema[ccs.constants.BTCCPRO]["ticker"]
```

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.

Key	Value	Description
Id	String	trade id.
Timestamp	UTCTimestamp	Unix time in seconds since 1 January 1970.
Price	Double	trade price
Quantity	Double	trade quantity
Side	Char	sell or buy

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.tradeHistory("XBTCNY")
>>> print(response)
[
    {
        "Id": 19,
        "Timestamp": 1456757388489,
        "Price": 2538,
        "Quantity": 2,
        "Side": "Sell"
    },
    ...
]
>>>
>>> # 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=10&limit=10&symbol=XBTCNY
- https://pro-data.btcc.com/data/pro/historydata?since=1456757387645&limit=10&sincetype=time&symbol=XBTCNY

CHAPTER 15

Btccusd

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

ccs.btccusd.public.**orderbook**(*symbol='BTCUSD'*, *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.

Key	Value	Description
asks	array	
bids	array	
date	number	last update timestamp

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

Position	Description
0	price
1	volume

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.btccusd.public.orderbook("BTCUSD")
>>> print(response)
{
    "asks":
            [
                [5721.48,0.8],
                [5721.4,0.71],
```

```
                        ...
                    ],
            "bids":
                    [
                        [5721,0.6097],
                        [5720.67,0.1],
                        ...
                    ],
            "date":1484398991
        }
        >>>
        >>> # Prepared validation schema
        >>> schema = ccs.cfg.schema[ccs.constants.BTCCUSD]["orderbook"]
```

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.

Key	Value	Description
BidPrice	Double	bid price
AskPrice	Double	ask pric
Open	Double	open price
High	Double	the highest trade price in 24 hours
Low	Double	the lowest trade price in 24 hours
Last	Double	last price
LastQuantity	Double	last quantity
PrevCls	Double	close Price
Timestamp	UTCTimestamp	timestamp
ExecutionLimitDown	Double	limit Down
ExecutionLimitUp	Double	limit Up

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.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
>>> schema = 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.

Key	Value	Description
Id	String	trade id.
Timestamp	UTCTimestamp	Unix time in seconds since 1 January 1970.
Price	Double	trade price
Quantity	Double	trade quantity
Side	Char	sell or buy

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.btccusd.public.tradeHistory("BTCUSD")
>>> print(response)
[
    {
        "Id": 19,
        "Timestamp": 1456757388489,
        "Price": 2538,
        "Quantity": 2,
        "Side": "Sell"
    },
    ...
]
>>>
>>> # Other examples of using
>>> 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
>>> schema = ccs.cfg.schema[ccs.constants.BTCCUSD]["tradeHistory"]
```

Note: This function use 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=10&limit=10
- https://spotusd-data.btcc.com/data/pro/historydata?symbol=BTCUSD&since=1456757387645&limit=10&sincetype=time

---

CHAPTER 16

Btce

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.

Key	Description
asks	sell orders
bids	buy orders

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

Position	Description
0	price
1	volume

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.btce.public.depth("btc_usd")
>>> print(response)
{
    "btc_usd":
            {
                "asks":
```

```
                                [
                                    [861.898,0.8],
                                    [861.899,0.00002518],
                                    ...
                                ],
                        "bids":
                                [
                                    [860,1.01],
                                    [859.203,0.37],
                                    ...
                                ]
                    }
            }
            >>>
            >>> # Other examples of using
            >>> ccs.btce.public.depth("btc_usd", limit=2)
            >>>
            >>> # Prepared validation schema
            >>> schema = ccs.cfg.schema[ccs.constants.BTCE]["depth"]
```

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.

Key	Description
decimal_places	number of decimals allowed during trading
min_price	minimum price allowed during trading
max_price	maximum price allowed during trading
min_amount	minimum sell or buy transaction size
hidden	whether the pair is hidden, 0 or 1
fee	commission for this pair

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.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
                        },
                        ...
            }
}
>>>
>>> # Prepared validation schema
>>> schema = ccs.cfg.schema[ccs.constants.BTCE]["info"]
```

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.

Key	Description
high	maximum price
low	minimum price
avg	average price
vol	trade volume
vol_cur	trade volume in currency
last	the price of the last trade
buy	buy price
sell	sell price
updated	last update of cache

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.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
            }
}
>>>
>>> # Prepared validation schema
>>> schema = ccs.cfg.schema[ccs.constants.BTCE]["ticker"]
```

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

`ccs.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 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.

Key	Description
type	ask – sell, bid – buy.
price	buy price or sell price.
amount	the amount of asset bought/sold.
tid	trade ID.
timestamp	Unix time of the trade.

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.btce.public.trades("btc_usd")
>>> 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
                },
                ...
            ]
}
>>>
>>> # Other examples of using
>>> ccs.btce.public.trades("btc_usd", limit=2)
>>>
>>> # Prepared validation schema
>>> schema = ccs.cfg.schema[ccs.constants.BTCE]["trades"]
```

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

Example of GET request:

•https://btc-e.com/api/3/trades/btc_usd

- https://btc-e.com/api/3/trades/btc_usd?limit=2

CHAPTER 17

Bter

17.1 Depth

ccs.bter.public.**depth**(*symbol*)

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

Key	Description
asks	
bids	

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

Position	Description
0	price
1	volume

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.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.

Key	Description
decimal_places	
min_amount	
fee	

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.bter.public.marketInfo()
            >>> print(response)
            {
                "result":"true",
                "pairs":
                        [
                            {
```

```
                    "btc_cny":
                            {
                                "decimal_places":2,
                                "min_amount":0.5,
                                "fee":0.2
                            }
                    },
                    {
                        "ltc_cny":
                            {
                                "decimal_places":2,
                                "min_amount":0.5,
                                "fee":0.2
                            }
                    },
                    ...
                ]
        }
        >>>
        >>> # Prepared validation schema
        >>> schema = ccs.cfg.schema[ccs.constants.BTER]["marketInfo"]
```

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.

Key	Description
no	
symbol	
name	
name_cn	
pair	
rate	
vol_a	
vol_b	
curr_a	
curr_b	
curr_suffix	
rate_percent	
trend	
supply	
marketcap	
plot	

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.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":"973,604",
                    "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
                        },
                        ...
                    ]
}
>>>
>>> # Prepared validation schema
>>> schema = ccs.cfg.schema[ccs.constants.BTER]["marketDetails"]
```

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.

Key	Description
result	
last	
high	
low	
avg	
sell	
buy	
vol_btc	
vol_cny	
rate_change_percentage	

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.bter.public.ticker("btc_cny")
>>> print(response)
{
     "result":"true",
     "last":6301.94,
     "high":6440,
     "low":6050,
     "avg":6250.29,
     "sell":6304.44,
     "buy":6302.95,
     "vol_btc":129.367,
     "vol_cny":808581.69,
     "rate_change_percentage":"-1.41"
}
>>>
>>> # 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.

Key	Description
result	
last	
high	
low	
avg	
sell	
buy	
vol_btc	
vol_cny	
rate_change_percentage	

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.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"
             },
             ...
}
>>>
>>> # Prepared validation schema
>>> schema = ccs.cfg.schema[ccs.constants.BTER]["tickers"]
```

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

Example of GET request:

- http://data.bter.com/api/1/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 traiding pair.

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.bter.public.tradingPairs()
>>> print(response)
[
    "btc_cny",
    "ltc_cny",
    "blk_cny",
    "bitcny_cny",
    "bqc_cny",
    "btb_cny",
    ...
]
>>>
>>> # Prepared validation schema
>>> schema = ccs.cfg.schema[ccs.constants.BTER]["tradingPairs"]
```

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

Example of GET request:

- http://data.bter.com/api/1/pairs

17.7 Trade history

ccs.bter.public.**tradeHistory**(*symbol*)

This function provide history of trades.

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 object. Official description of keys is in the table.

Key	Description
date	
price	
tid	
type	

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.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"
}
>>>
>>> # Prepared validation schema
>>> schema = ccs.cfg.schema[ccs.constants.BTER]["tradeHistory"]
```

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

CHAPTER 18

Cexio

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.

Key	Description
tmsp	UNIX timestamp
price	price value

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.cexio.public.chart("BTC", "USD", 24, 100)
>>> print(response)
[
    {
        "tmsp":1482246000,
        "price":"796.658"
    },
    {
```

```
            "tmsp":1482246900,
            "price":"796.1659"
        },
        ...
]
>>>
>>> # Prepared validation schema
>>> schema = ccs.cfg.schema[ccs.constants.CEXIO]["chart"]
```

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.

Key	Description
amnt	amount in the target currency

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

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

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

Here is not example, because this request is executed by POST method.

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.

Key	Description
symbol1	?
symbol2	?
minLotSize	?
minLotSizeS2	?
maxLotSize	?
minPrice	?
maxPrice	?

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.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,
```

```
                                        "minLotSizeS2":2.2,
                                        "maxLotSize":30,
                                        "minPrice":"100.00",
                                        "maxPrice":"1900.00"
                                    },
                                    ...
                                ]
                        }
                }
        }
        >>>
        >>> # Prepared validation schema
        >>> schema = ccs.cfg.schema[ccs.constants.CEXIO]["currencyLimits"]
```

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.

Key	Description
time	Unix timestamp
data1m	1440 candle data sets

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

Position	Description
0	Unix timestamp
1	open price
2	high price
3	low price
4	close price
5	volume

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.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.

Key	Description
curr1	the first currency code;
curr2	the second currency code;
lprice	last price of selling/buying the first currency relative to the second one

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.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.

Key	Description
symbol1	the first currency code;
symbol2	the second currency code;
lprice	last price of selling/buying the first currency relative to the second one

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.cexio.public.lastPricesForGivenMarket("BTC", "USD", "LTC")
>>> print(response)
{
     "e":"last_prices",
     "ok":"ok",
     "data":
             [
                 {
                     "symbol1":"BTC",
                     "symbol2":"USD",
                     "lprice":"937.545"
                 },
                 {
                     "symbol1":"LTC",
                     "symbol2":"USD",
                     "lprice":"4.0129"
                 },
                 ...
             ]
}
>>>
>>> # Prepared validation schema
>>> schema = ccs.cfg.schema[ccs.constants.CEXIO]["lastPricesForGivenMarket"]
```

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.

Key	Description
timestamp	Unix timestamp
bids	lists of open orders, each order is represented as a list
asks	lists of open orders, each order is represented as a list
pair	pair name
id	incremental version id of order-book snapshot, may be used to check changes
sell_total	total available in symbol1 (cur1)
buy_total	total available in symbol2 (cur2)

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

Position	Description
0	price
1	volume

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

•https://cex.io/api/order_book/BTC/USD/

•https://cex.io/api/order_book/BTC/USD/?depth=1

18.8 Ticker

ccs.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.

Key	Description
timestamp	unix timestamp
low	last 24 hours price low
high	last 24 hours price high
last	last BTC price
volume	last 24 hours volume
volume30d	last 30 days volume
bid	highest buy order
ask	lowest sell order

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.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
}
>>>
```

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

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

Example of GET request:

•https://cex.io/api/ticker/BTC/USD

18.9 Tickers for all pairs by market

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.

Key	Description
timestamp	unix timestamp
low	last 24 hours price low
high	last 24 hours price high
last	last BTC price
volume	last 24 hours volume
volume30d	last 30 days volume
bid	highest buy order
ask	lowest sell order

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.cexio.public.tickersForAllPairsByMarket("USD", "EUR", "RUB", "BTC")
>>> print(response)
{
     "e":"tickers",
     "ok":"ok",
     "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.

Key	Description
tid	unique trade id
type	buy or sell
amount	trade amount
price	price
date	Unix timestamp

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.cexio.public.tradeHistory("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"
    },
    ...
]
>>>
>>> # Other examples of using
>>> ccs.cexio.public.tradeHistory("BTC", "USD", since=1)
>>>
>>> # 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

CHAPTER 19

Kraken

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.

Key	Description
altname	alternate name
aclass	asset class
decimals	scaling decimal places for record keeping
display_decimals	scaling decimal places for output display

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.kraken.public.getAssetInfo()
>>> print(response)
{
    "error":[],
    "result":
                {
                    "KFEE":
                            {
                                "aclass":"currency",
                                "altname":"FEE",
                                "decimals":2,
                                "display_decimals":2
                            },
                    "XDAO":
                            {
                                "aclass":"currency",
                                "altname":"DAO",
                                "decimals":10,
                                "display_decimals":3
                            },
                            ...
                }
}
>>>
>>> # 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
- https://api.kraken.com/0/public/Assets?asset=ZEUR
- https://api.kraken.com/0/public/Assets?asset=ZEUR,XXBT
- https://api.kraken.com/0/public/Assets?aclass=currency

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.

Value	Description
info	all info (default)
leverage	leverage info
fees	fees schedule
margin	margin info

- **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.

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

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.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:

•https://api.kraken.com/0/public/AssetPairs

•https://api.kraken.com/0/public/AssetPairs?pair=XXBTZEUR

•https://api.kraken.com/0/public/AssetPairs?pair=XXBTZEUR&info=leverage

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.

Position	Description
0	time
1	open
2	high
3	low
4	close
5	vwap
6	volume
7	count

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

Example

```
>>> 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
                        }
        }
        >>>
        >>> # Other examples of using
        >>> ccs.kraken.public.getOHLCdata("XBTEUR", interval=5)
        >>> ccs.kraken.public.getOHLCdata("XBTEUR", since=1482689400)
        >>> ccs.kraken.public.getOHLCdata("XBTEUR", interval=5, since=1482689400)
        >>> # Prepared validation schema
        >>> schema = ccs.cfg.schema[ccs.constants.KRAKEN]["getOHLCdata"]
```

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&interval=5
- 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** (*Interger*) – 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.

Position	Description
0	price
1	volume
2	timestamp

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.kraken.public.getOrderBook("XBTEUR")
>>> print(response)
{
    "error":[],
    "result":
            {
                "XXBTZEUR":
                            {
                                "asks":
                                        [
                                            ["863.24000","1.753",1482580426],
                                            ["863.61000","12.500",14825797746],
                                            ...
                                        ],
                                "bids":
                                        [
                                            ["862.00000","0.001",1482580604],
                                            ["861.48000","3.198",1482580657],
                                            ...
                                        ]
                            }
            }
}
>>>
>>> # Other examples of using
>>> ccs.kraken.public.getOrderBook("XBTEUR", 3)
>>>
>>> # Prepared validation schema
>>> schema = ccs.cfg.schema[ccs.constants.KRAKEN]["getOrderBook"]
```

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

Example of GET request:

- https://api.kraken.com/0/public/Depth?pair=XBTEUR
- https://api.kraken.com/0/public/Depth?pair=XBTEUR&count=2

19.5 Ticker

ccs.kraken.public.**getTickerInformation**(*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*) – It is currency pair. For more information about symbols visit *getTradableAssetPairs()*.

Returns

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

Key	Description
a	ask array(*price*, *whole lot volume*, *lot volume*)
b	bid array(*price*, *whole lot volume*, *lot volume*)
c	last trade closed array(*price*, *lot volume*)
v	volume array(*today*, *last 24 hours*),
p	volume weighted average price array(*today*, *last 24 hours*)
t	number of trades array(*today*, *last 24 hours*)
l	low array(*today*, *last 24 hours*)
h	high array(*today*, *last 24 hours*)
o	today's opening price

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.kraken.public.getTickerInformation("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"
                            }
            }
}
```

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

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.

Position	Description
0	price
1	volume
2	time
3	buy / sell
4	market / limit
5	miscellaneous

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

```
>>> import ccs
>>> response = ccs.kraken.public.getRecentTrades("XBTEUR")
>>> print(response)
{
    "error":[],
    "result":
            {
                "XXBTZEUR":
                            [
```

```
                                        ["863.73500","0.02750313",1482576757.9252,"s","m",""],
                                        ["863.73500","4.03892266",1482576797.757,"s","l",""],
                                        ...
                                    ],
                            "last":"1482576827813240845"
                        }
            }
            >>>
            >>> # Other examples of using
            >>> ccs.kraken.public.getRecentTrades("XBTEUR", 1482576757925126325)
            >>>
            >>> # Prepared validation schema
            >>> 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
- https://api.kraken.com/0/public/Trades?pair=XBTEUR&since=1482576757925126325

19.7 Server time

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

```
            >>> import ccs
            >>> response = ccs.kraken.public.getServerTime()
            >>> print(response)
            {
                "error":[],
                "result":
                        {
                            "unixtime":1482674808,
                            "rfc1123":"Sun, 25 Dec 16 14:06:48 +0000"
                        }
            }
            >>>
            >>> # Prepared validation schema
            >>> schema = ccs.cfg.schema[ccs.constants.KRAKEN]["getServerTime"]
```

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

Example of GET request:

- 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.

Position	Description
0	time
1	bid
2	ask

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

```
>>> 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
            }
```

```
            }
            >>>
            >>> # Other examples of using
            >>> ccs.kraken.public.getRecentSpreadData("XBTEUR", since=1482690474)
            >>> # Prepared validation schema
            >>> schema = ccs.cfg.schema[ccs.constants.KRAKEN]["getRecentSpreadData"]
```

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

Example of GET request:

-

CHAPTER 20

Okcoin

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.

Key	Description
asks	ask depth
bids	bid depth

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

Position	Description
0	price
1	volume

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.okcoincom.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.okcoincn.public.depth("btc_cny")
        >>> ccs.okcoincom.public.depth("btc_usd", size=2)
        >>> ccs.okcoincom.public.depth("btc_usd", merge=1)
        >>> ccs.okcoincom.public.depth("btc_usd", size=2, merge=1)
        >>>
        >>> # Prepared validation schema
        >>> schema = ccs.cfg.schema[ccs.constants.OKCOINCOM]["depth"]
        >>> schema = ccs.cfg.schema[ccs.constants.OKCOINCN]["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
- https://www.okcoin.com/api/v1/depth.do?symbol=btc_usd&size=2
- 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.

Position	Description
0	timestamp
1	open
2	high
3	low
4	close
5	volume

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

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

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

Example of GET request:

- 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.

Key	Description
date	server time for returned data
buy	best bid
high	highest price
last	latest price
low	lowest price
sell	best ask
vol	volume (in the last rolling 24 hours)

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.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"
            }
}
>>>
>>> # 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.com/api/v1/ticker.do?symbol=btc_usd
- https://www.okcoin.com/api/v1/ticker.do?symbol=ltc_usd
- https://www.okcoin.cn/api/v1/ticker.do?symbol=btc_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.

Key	Description
date	transaction time
date_ms	transaction time in milliseconds
price	transaction price
amount	quantity in BTC (or LTC)
tid	transaction ID
type	buy/sell

rtype String

exception It can raise any exception which can occur during using

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

Example

```
>>> 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=btc_usd
- 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

CHAPTER 21

Poloniex

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.

Key	Unofficial description
id	TODO
last	current price
lowestAsk	current lowest ask
highestBid	current highest bid
percentChange	percent change of price
baseVolume	volume of trades in base currency
quoteVolume	volume of trades in quote currency
isFrozen	TODO
high24hr	TODO
low24hr	TODO

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.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"
                    },
                    ...
    }
    >>>
    >>> # Prepared validation schema
    >>> schema = ccs.cfg.schema[ccs.constants.POLONIEX]["returnTicker"]
```

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

Example of GET request:

- https://poloniex.com/public?command=returnTicker

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 recomandation: 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 recomandation: 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

```
>>> 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"
    },
    ...
]
>>>
>>> # Other examples of using
>>> ccs.poloniex.public.returnTradeHistory("BTC_LTC", start=1410158341, end=1410499372)
>>>
>>> # Prepared validation schema
>>> schema = ccs.cfg.schema[ccs.constants.POLONIEX]["returnTradeHistory"]
```

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

Example of GET request:

- https://poloniex.com/public?command=returnTradeHistory&currencyPair=BTC_NXT
- https://poloniex.com/public?command=returnTradeHistory&currencyPair=BTC_NXT&start=1410158341&end=14104993

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.

Key	Description
asks	list of orders
bids	list of orders
isFrozen	?
seq	?

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

Position	Description
0	price
1	volume

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.poloniex.public.returnOrderBook("BTC_LTC")
>>> 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
>>> ccs.poloniex.public.returnOrderBook("BTC_NXT", 30)
>>> ccs.poloniex.public.returnOrderBook("all")
>>>
>>> # Prepared validation schema
>>> schema = ccs.cfg.schema[ccs.constants.POLONIEX]["returnOrderBook"]
```

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

Example of GET request:

- https://poloniex.com/public?command=returnOrderBook&currencyPair=BTC_NXT&depth=10
- https://poloniex.com/public?command=returnOrderBook&currencyPair=all&depth=2

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 escription of object's keys is in the table.

Key	Description
<base>_<quote>	Json object contains 24 hours volumes for base and quote currency of pair.
total<base>	Sum of volumes for base currency in last 24 hours.

<*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

```
>>> import ccs
>>> response = ccs.poloniex.public.return24hVolume()
>>> print(response)
{
     "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:

- https://poloniex.com/public?command=return24hVolume

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 recomandation: 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 recomandation: 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.

Key	Description
date	unix timestamp
high	candle attribute - higher price in period
low	candle attribute - lower price in period
open	candle attribute - opening price in period
close	candle attribute - closing price in period
volume	volume of base currency in period
quoteVolume	volume of quote currency in period
weightedAverage	weighted average in period

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.poloniex.public.returnChartData("BTC_LTC", 1405699200, 9999999999, 30
>>> 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":0.01413455
     },
     ...
 ]
>>>
>>> # Prepared validation schema
>>> schema = ccs.cfg.schema[ccs.constants.POLONIEX]["returnChartData"]
```

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

Example of GET request:

•https://poloniex.com/public?command=returnChartData&currencyPair=BTC_XMR&start=1405699200&end=9999999999

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.

Key	Description
id	unique ID
name	full name of currency
txFee	fee
minConf	?
depositAddress	?
disabled	?
delisted	?
frozen	?

rtype String

exception It can raise any exception which can occur during using

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

Example

```
>>> 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
>>> schema = ccs.cfg.schema[ccs.constants.POLONIEX]["returnCurrencies"]
```

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

Example of GET request:

- https://poloniex.com/public?command=returnCurrencies

21.7 Loan orders

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.

Key	Description
offers	
demands	

IMPROVE

Key	Description
rate	
amount	
rangeMin	
rangeMax	

rtype String

exception It can raise any exception which can occur during using

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

Example

```
>>> import ccs
>>> response = ccs.poloniex.public.returnLoanOrders("BTC")
>>> print(response)
{
    "offers":
            [
                {
                    "rate":"0.00018500",
                    "amount":"0.01487170",
                    "rangeMin":2,
                    "rangeMax":2
                },
```

```
                        {
                            "rate":"0.00018599",
                            "amount":"0.47963188",
                            "rangeMin":2,
                            "rangeMax":2
                        },
                        ...
                    ],
            "demands":
                    [
                        {
                            "rate":"0.00012100",
                            "amount":"28.62300354",
                            "rangeMin":2,
                            "rangeMax":2
                        },
                        {
                            "rate":"0.00012000",
                            "amount":"54.51656874",
                            "rangeMin":2,
                            "rangeMax":2
                        },
                        ...
                    ]
        }
        >>>
        >>> # Prepared validation schema
        >>> schema = ccs.cfg.schema[ccs.constants.POLONIEX]["returnLoanOrders"]
```

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

Example of GET request:

- https://poloniex.com/public?command=returnLoanOrders&currency=BTC

CHAPTER 22

Indices and tables

- genindex
- modindex
- search

C

S

T