ccxt Documentation

Release 1.17.235

Igor Kroitor

Sep 06, 2018

General Information

1 Supported Exchanges **3**

2 Exchanges By Country **11**

3 Install **21**
3.1 JavaScript (NPM) . . . 21
3.2 JavaScript (for use with the <script> tag): . . . 22
3.3 Python . . . 22
3.4 PHP . . . 22

4 Proxy **25**
4.1 JavaScript Proxies . . . 25
4.2 Python Proxies . . . 25

5 CORS (Access-Control-Allow-Origin) **29**
5.1 Node.js CORS Proxy . . . 29
5.2 Python CORS Proxy . . . 30
5.3 Testing CORS . . . 30

6 Overview **31**

7 Exchanges **33**
7.1 Instantiation . . . 39
7.2 Exchange Structure . . . 41
7.3 Rate Limit . . . 44

8 Markets **47**
8.1 Market Structure . . . 47
8.2 Loading Markets . . . 49
8.3 Symbols And Market Ids . . . 49
8.4 Market Cache Force Reload . . . 53

9 API Methods / Endpoints **55**
9.1 Implicit API Methods . . . 55
9.2 Public/Private API . . . 56
9.3 Synchronous vs Asynchronous Calls . . . 57
9.4 Returned JSON Objects . . . 58
9.5 Passing Parameters To API Methods . . . 58

9.6 Unified API . . . 58

10 Market Data **65**
10.1 Order Book . . . 65
10.2 Price Tickers . . . 68
10.3 OHLCV Candlestick Charts . . . 71
10.4 Trades, Executions, Transactions . . . 72

11 Trading **75**
11.1 Authentication . . . 75
11.2 API Keys Setup . . . 76
11.3 Querying Account Balance . . . 77
11.4 Orders . . . 79
11.5 Personal Trades . . . 87
11.6 Funding Your Account . . . 90
11.7 Fees . . . 93
11.8 Overriding The Nonce . . . 95

12 Error Handling **97**
12.1 ExchangeError . . . 98
12.2 NetworkError . . . 99

13 Troubleshooting **101**
13.1 Notes . . . 102

14 Frequently Asked Questions **103**
14.1 I'm trying to run the code, but it's not working, how do I fix it? . . . 103
14.2 I am calling a method and I get an error, what am I doing wrong? . . . 103
14.3 I got an incorrect result from a method call, can you help? . . . 104
14.4 Can you implement feature `foo` in exchange `bar`? . . . 104
14.5 When will you add feature `foo` for exchange `bar` ? What's the estimated time? When should we expect this? . . . 104
14.6 What's your progress on adding the feature `foo` that was requested earlier? How do you do implementing exchange `bar`? . . . 104
14.7 Hey! The fix you've uploaded is in JS, would you fix Python / PHP as well, please? . . . 104

15 CCXT – CryptoCurrency eXchange Trading Library **105**
15.1 Install · Usage · Manual · FAQ · Examples · Contributing · Social . . . 105
15.2 JavaScript (NPM) . . . 113
15.3 JavaScript (for use with the `<script>` tag): . . . 113
15.4 Python . . . 113
15.5 PHP . . . 114
15.6 Intro . . . 114
15.7 JavaScript . . . 115
15.8 Python . . . 116
15.9 PHP . . . 117
15.10 Sponsors . . . 118
15.11 Backers . . . 118
15.12 Crypto . . . 118

16 Supported Cryptocurrency Exchange Markets **121**

17 Install **127**
17.1 JavaScript (NPM) . . . 127
17.2 JavaScript (for use with the `<script>` tag): . . . 128

17.3 Python . . . 128
17.4 PHP . . . 128

18 Documentation **129**

19 Usage **131**
19.1 Intro . . . 131
19.2 JavaScript . . . 132
19.3 Python . . . 133
19.4 PHP . . . 133

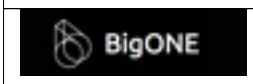
20 Contributing **135**

A JavaScript / Python / PHP library for cryptocurrency trading and e-commerce with support for many bitcoin/ether/altcoin exchange markets and merchant APIs.

CHAPTER 1

Supported Exchanges

The ccxt library currently supports the following 131 cryptocurrency exchange markets and trading APIs:

	id	name	certified	ver	doc	countries
	_1broker	1Broker		2	API	US
	_1btcxe	1BTCXE		*	API	Panama
	acx	ACX		2	API	Australia
	allcoin	Allcoin		1	API	Canada
	anxpro	ANXPro		2	API	Japan, Singapore, Hong Kon
	anybits	Anybits		*	API	Ireland
	bcex	BCEX		1	API	China, Canada
	bibox	Bibox		1	API	China, US, South Korea
	bigone	BigONE		2	API	UK
	binance	Binance	CCXT Certified	*	API	Japan
	bit2c	Bit2C		*	API	Israel

Continue

Table 1 – continued from previous page

	id	name	certified	ver	doc	countries
	bitbank	bitbank		1	API	Japan
	bitbay	BitBay		*	API	Malta, EU
	bitfinex	Bitfinex	CCXT Certified	1	API	British Virgin Islands
	bitfinex2	Bitfinex v2		2	API	British Virgin Islands
	bitflyer	bitFlyer		1	API	Japan
	bitforex	Bitforex		1	API	China
	bithumb	Bithumb		*	API	South Korea
	bitkk	bitkk		1	API	China
	bitlish	Bitlish		1	API	UK, EU, Russia
	bitmarket	BitMarket		*	API	Poland, EU
	bitmex	BitMEX		1	API	Seychelles
	bitsane	Bitsane		*	API	Ireland
	bitso	Bitso		3	API	Mexico
	bitstamp	Bitstamp		2	API	UK
	bitstamp1	Bitstamp v1		1	API	UK
	bittrex	Bittrex	CCXT Certified	1.1	API	US
	bitz	Bit-Z		2	API	Hong Kong
	bl3p	BL3P		1	API	Netherlands, EU
	bleutrade	Bleutrade		2	API	Brazil
	braziliex	Braziliex		*	API	Brazil
	btcalpha	BTC-Alpha		1	API	US

Continued

Table 1 – continued from previous page

	id	name	certified	ver	doc	countries
	btcbox	BtcBox		1	API	Japan
	btcchina	BTCChina		1	API	China
	btcexchange	BTCExchange		*	API	Philippines
	btcmarkets	BTC Markets		*	API	Australia
	btctradeim	BtcTrade.im		*	API	Hong Kong
	btctradeua	BTC Trade UA		*	API	Ukraine
	btcturk	BTCTurk		*	API	Turkey
	btcx	BTCX		1	API	Iceland, US, EU
	bxinth	BX.in.th		*	API	Thailand
	ccex	C-CEX		*	API	Germany, EU
	cex	CEX.IO		*	API	UK, EU, Cyprus, Russia
	chbtc	CHBTC		1	API	China
	chilebit	ChileBit		1	API	Chile
	cobinhood	COBINHOOD		1	API	Taiwan
	coinbase	coinbase		2	API	US
	coinbaseprime	Coinbase Prime		*	API	US
	coinbasepro	Coinbase Pro		*	API	US
	coincheck	coincheck		*	API	Japan, Indonesia
	coinegg	CoinEgg		*	API	China, UK
	coinex	CoinEx		1	API	China
	coinexchange	CoinExchange		*	API	India, Japan, South Korea, V

Continue

Table 1 – continued from previous page

	id	name	certified	ver	doc	countries
	coinfalcon	CoinFalcon		1	API	UK
	coinfloor	coinfloor		*	API	UK
	coingi	Coingi		*	API	Panama, Bulgaria, China, US
	coinmarketcap	CoinMarketCap		1	API	US
	coinmate	CoinMate		*	API	UK, Czech Republic, EU
	coinnest	coinnest		*	API	South Korea
	coinone	CoinOne		2	API	South Korea
	coinsecure	Coinsecure		1	API	India
	coinspot	CoinSpot		*	API	Australia
	cointiger	CoinTiger		1	API	China
	coolcoin	CoolCoin		*	API	Hong Kong
	crypton	Crypton		1	API	EU
	cryptopia	Cryptopia		*	API	New Zealand
	deribit	Deribit		1	API	Netherlands
	dsx	DSX		3	API	UK
	ethfinex	Ethfinex		1	API	British Virgin Islands
	exmo	EXMO		1	API	Spain, Russia
	exx	EXX		*	API	China
	fcoin	FCoin		2	API	China
	flowbtc	flowBTC		1	API	Brazil
	foxbit	FoxBit		1	API	Brazil

Continued

Table 1 – continued from previous page

	id	name	certified	ver	doc	countries
	fybse	FYB-SE		*	API	Sweden
	fybsg	FYB-SG		*	API	Singapore
	gatecoin	Gatecoin		*	API	Hong Kong
	gateio	Gate.io		2	API	China
	gdax	GDAX		*	API	US
	gemini	Gemini		1	API	US
	getbtc	GetBTC		*	API	St. Vincent & Grenadines, R
	hadax	HADAX		1	API	China
	hitbtc	HitBTC		1	API	Hong Kong
	hitbtc2	HitBTC v2		2	API	Hong Kong
	huobi	Huobi		3	API	China
	huobicny	Huobi CNY		1	API	China
	huobipro	Huobi Pro		1	API	China
	ice3x	ICE3X		1	API	South Africa
	independentreserve	Independent Reserve		*	API	Australia, New Zealand
	indodax	INDODAX		1.8	API	Indonesia
	itbit	itBit		1	API	US
	jubi	jubi.com		1	API	China
	kraken	Kraken	CCXT Certified	0	API	US
	kucoin	Kucoin		1	API	Hong Kong
	kuna	Kuna		2	API	Ukraine

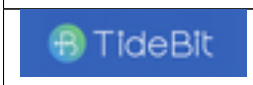
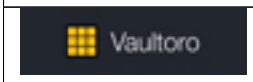
Continue

Table 1 – continued from previous page

	id	name	certified	ver	doc	countries
	lakebtc	LakeBTC		2	API	US
	lbank	LBank		1	API	China
	liqui	Liqui		3	API	Ukraine
	livecoin	LiveCoin		*	API	US, UK, Russia
	luno	luno		1	API	UK, Singapore, South Africa
	lykke	Lykke		1	API	Switzerland
	mercado	Mercado Bitcoin		3	API	Brazil
	mixcoins	MixCoins		1	API	UK, Hong Kong
	negociecoins	NegocieCoins		3	API	Brazil
	nova	Novaexchange		2	API	Tanzania
	okcoincny	OKCoin CNY		1	API	China
	okcoinusd	OKCoin USD		1	API	China, US
	okex	OKEX		1	API	China, US
	paymium	Paymium		1	API	France, EU
	poloniex	Poloniex		*	API	US
	qryptos	QRYPTOS		2	API	China, Taiwan
	quadrigacx	QuadrigaCX		2	API	Canada
	quoinex	QUOINEX		2	API	Japan, Singapore, Vietnam
	rightbtc	RightBTC		*	API	United Arab Emirates
	southxchange	SouthXchange		*	API	Argentina
	surbitcoin	SurBitcoin		1	API	Venezuela

Continued

Table 1 – continued from previous page

	id	name	certified	ver	doc	countries
	theocean	The Ocean	CCXT Certified	0	API	US
	therock	TheRockTrading		1	API	Malta
	tidebit	TideBit		2	API	Hong Kong
	tidex	Tidex		3	API	UK
	uex	UEX		1.0.3	API	Singapore, US
	urdubit	UrduBit		1	API	Pakistan
	vaultoro	Vaultoro		1	API	Switzerland
	vbtc	VBTC		1	API	Vietnam
	virwox	VirWoX		*	API	Austria, EU
	wex	WEX		3	API	New Zealand
	xbtce	xBTCe		1	API	Russia
	yobit	YoBit		3	API	Russia
	yunbi	YUNBI		2	API	China
	zaif	Zaif		1	API	Japan
	zb	ZB		1	API	China

CHAPTER 2

Exchanges By Country

The ccxt library currently supports the following cryptocurrency exchange markets and trading APIs:

country / region	logo	id	name	certified	ver	doc
Argentina		southxchange	SouthXchange		*	API
Australia		acx	ACX		2	API
Australia		btcmarkets	BTC Markets		*	API
Australia		coinspot	CoinSpot		*	API
Australia		independentreserve	Independent Reserve		*	API
Austria		virwox	VirWoX		*	API
Brazil		bleutrade	Bleutrade		2	API
Brazil		braziliex	Braziliex		*	API
Brazil		flowbtc	flowBTC		1	API
Brazil		foxbit	FoxBit		1	API
Brazil		mercado	Mercado Bitcoin		3	API

Continued on next page

Table 1 – continued from previous page

country / region	logo	id	name	certified	ver	doc
Brazil		negociecoins	NegocieCoins		3	API
British Virgin Islands		bitfinex	Bitfinex	CCXT Certified	1	API
British Virgin Islands		bitfinex2	Bitfinex v2		2	API
British Virgin Islands		ethfinex	Ethfinex		1	API
Bulgaria		coingi	Coingi		*	API
Canada		allcoin	Allcoin		1	API
Canada		bcex	BCEX		1	API
Canada		quadrigacx	QuadrigaCX		2	API
Chile		chilebit	ChileBit		1	API
China		bcex	BCEX		1	API
China		bibox	Bibox		1	API
China		bitforex	Bitforex		1	API
China		bitkk	bitkk		1	API
China		btcchina	BTCChina		1	API
China		chbtc	CHBTC		1	API
China		coinegg	CoinEgg		*	API
China		coinex	CoinEx		1	API
China		coingi	Coingi		*	API
China		cointiger	CoinTiger		1	API
China		exx	EXX		*	API
China		fcoin	FCoin		2	API

Continued on next page

Table 1 – continued from previous page

country / region	logo	id	name	certified	ver	doc
China		gateio	Gate.io		2	API
China		hadax	HADAX		1	API
China		huobi	Huobi		3	API
China		huobicny	Huobi CNY		1	API
China		huobipro	Huobi Pro		1	API
China		jubi	jubi.com		1	API
China		lbank	LBank		1	API
China		okcoincny	OKCoin CNY		1	API
China		okcoinusd	OKCoin USD		1	API
China		okex	OKEX		1	API
China		qryptos	QRYPTOS		2	API
China		yunbi	YUNBI		2	API
China		zb	ZB		1	API
Cyprus		cex	CEX.IO		*	API
Czech Republic		coinmate	CoinMate		*	API
EU		bitbay	BitBay		*	API
EU		bitlish	Bitlish		1	API
EU		bitmarket	BitMarket		*	API
EU		bl3p	BL3P		1	API
EU		btcx	BTCX		1	API
EU		ccex	C-CEX		*	API

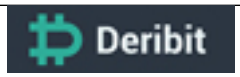
Continued on next page

Table 1 – continued from previous page

country / region	logo	id	name	certified	ver	doc
EU		cex	CEX.IO		*	API
EU		coinmate	CoinMate		*	API
EU		crypton	Crypton		1	API
EU		paymium	Paymium		1	API
EU		virwox	VirWoX		*	API
France		paymium	Paymium		1	API
Germany		ccex	C-CEX		*	API
Hong Kong		anxpro	ANXPro		2	API
Hong Kong		bitz	Bit-Z		2	API
Hong Kong		btctradeim	BtcTrade.im		*	API
Hong Kong		coolcoin	CoolCoin		*	API
Hong Kong		gatecoin	Gatecoin		*	API
Hong Kong		hitbtc	HitBTC		1	API
Hong Kong		hitbtc2	HitBTC v2		2	API
Hong Kong		kucoin	Kucoin		1	API
Hong Kong		mixcoins	MixCoins		1	API
Hong Kong		tidebit	TideBit		2	API
Iceland		btcx	BTCX		1	API
India		coinexchange	CoinExchange		*	API
India		coinsecure	Coinsecure		1	API
Indonesia		coincheck	coincheck		*	API

Continued on next page

Table 1 – continued from previous page

country / region	logo	id	name	certified	ver	doc
Indonesia		indodax	INDODAX		1.8	API
Ireland		anybits	Anybits		*	API
Ireland		bitsane	Bitsane		*	API
Israel		bit2c	Bit2C		*	API
Japan		anxpro	ANXPro		2	API
Japan		binance	Binance	CCXT Certified	*	API
Japan		bitbank	bitbank		1	API
Japan		bitflyer	bitFlyer		1	API
Japan		btcbox	BtcBox		1	API
Japan		coincheck	coincheck		*	API
Japan		coinexchange	CoinExchange		*	API
Japan		quoinex	QUOINEX		2	API
Japan		zaif	Zaif		1	API
Malta		bitbay	BitBay		*	API
Malta		therock	TheRockTrading		1	API
Mexico		bitso	Bitso		3	API
Netherlands		bl3p	BL3P		1	API
Netherlands		deribit	Deribit		1	API
New Zealand		anxpro	ANXPro		2	API
New Zealand		cryptopia	Cryptopia		*	API
New Zealand		independentreserve	Independent Reserve		*	API

Continued on next page

Table 1 – continued from previous page

country / region	logo	id	name	certified	ver	doc
New Zealand		wex	WEX		3	API
Pakistan		urdubit	UrduBit		1	API
Panama		_1btcxe	1BTCXE		*	API
Panama		coingi	Coingi		*	API
Philippines		btcexchange	BTCExchange		*	API
Poland		bitmarket	BitMarket		*	API
Russia		bitlish	Bitlish		1	API
Russia		cex	CEX.IO		*	API
Russia		exmo	EXMO		1	API
Russia		getbtc	GetBTC		*	API
Russia		livecoin	LiveCoin		*	API
Russia		xbtce	xBTCe		1	API
Russia		yobit	YoBit		3	API
Seychelles		bitmex	BitMEX		1	API
Singapore		anxpro	ANXPro		2	API
Singapore		fybsg	FYB-SG		*	API
Singapore		luno	luno		1	API
Singapore		quoinex	QUOINEX		2	API
Singapore		uex	UEX		1.0.3	API
South Africa		ice3x	ICE3X		1	API
South Africa		luno	luno		1	API

Continued on next page

Table 1 – continued from previous page

country / region	logo	id	name	certified	ver	doc
South Korea		bibox	Bibox		1	API
South Korea		bithumb	Bithumb		*	API
South Korea		coinexchange	CoinExchange		*	API
South Korea		coinnest	coinnest		*	API
South Korea		coinone	CoinOne		2	API
Spain		exmo	EXMO		1	API
St. Vincent & Grenadines		getbtc	GetBTC		*	API
Sweden		fybse	FYB-SE		*	API
Switzerland		lykke	Lykke		1	API
Switzerland		vaultoro	Vaultoro		1	API
Taiwan		cobinhood	COBINHOOD		1	API
Taiwan		qryptos	QRYPTOS		2	API
Tanzania		nova	Novaexchange		2	API
Thailand		bxinth	BX.in.th		*	API
Turkey		btcturk	BTCTurk		*	API
UK		bigone	BigONE		2	API
UK		bitlish	Bitlish		1	API
UK		bitstamp	Bitstamp		2	API
UK		bitstamp1	Bitstamp v1		1	API
UK		cex	CEX.IO		*	API
UK		coinegg	CoinEgg		*	API

Continued on next page

Table 1 – continued from previous page

country / region	logo	id	name	certified	ver	doc
UK		coinfalcon	CoinFalcon		1	API
UK		coinfloor	coinfloor		*	API
UK		coinmate	CoinMate		*	API
UK		dsx	DSX		3	API
UK		livecoin	LiveCoin		*	API
UK		luno	luno		1	API
UK		mixcoins	MixCoins		1	API
UK		tidex	Tidex		3	API
Ukraine		btctradeua	BTC Trade UA		*	API
Ukraine		kuna	Kuna		2	API
Ukraine		liqui	Liqui		3	API
United Arab Emirates		rightbtc	RightBTC		*	API
US		_1broker	1Broker		2	API
US		bibox	Bibox		1	API
US		bittrex	Bittrex	CCXT Certified	1.1	API
US		btcalpha	BTC-Alpha		1	API
US		btcx	BTCX		1	API
US		coinbase	coinbase		2	API
US		coinbaseprime	Coinbase Prime		*	API
US		coinbasepro	Coinbase Pro		*	API
US		coinexchange	CoinExchange		*	API

Continued on next page

Table 1 – continued from previous page

country / region	logo	id	name	certified	ver	doc
US		coingi	Coingi		*	API
US		coinmarketcap	CoinMarketCap		1	API
US		gdax	GDAX		*	API
US		gemini	Gemini		1	API
US		itbit	itBit		1	API
US		kraken	Kraken	CCXT Certified	0	API
US		lakebtc	LakeBTC		2	API
US		livecoin	LiveCoin		*	API
US		okcoinusd	OKCoin USD		1	API
US		okex	OKEX		1	API
US		poloniex	Poloniex		*	API
US		theocean	The Ocean	CCXT Certified	0	API
US		uex	UEX		1.0.3	API
Venezuela		surbitcoin	SurBitcoin		1	API
Vietnam		coinexchange	CoinExchange		*	API
Vietnam		quoinex	QUOINEX		2	API
Vietnam		vbtc	VBTC		1	API

CHAPTER 3

Install

The easiest way to install the ccxt library is to use builtin package managers:

- ccxt in NPM (JavaScript / Node v7.6+)
- ccxt in PyPI (Python 2 and 3)

This library is shipped as an all-in-one module implementation with minimalistic dependencies and requirements:

- `ccxt.js <https://github.com/ccxt/ccxt/blob/master/ccxt.js>`__ in JavaScript
- `./python/ <https://github.com/ccxt/ccxt/blob/master/python/>`__ in Python (generated from JS)
- `ccxt.php <https://github.com/ccxt/ccxt/blob/master/ccxt.php>`__ in PHP (generated from JS)

You can also clone it into your project directory from ccxt GitHub repository and copy files manually into your working directory with language extension appropriate for your environment.

```
git clone https://github.com/ccxt/ccxt.git
```

An alternative way of installing this library is to build a custom bundle from source. Choose exchanges you need in `exchanges.cfg`.

3.1 JavaScript (NPM)

JavaScript version of ccxt works both in Node and web browsers. Requires ES6 and `async/await` syntax support (Node 7.6.0+). When compiling with Webpack and Babel, make sure it is not excluded in your `babel-loader` config.

ccxt crypto trading library in npm

```
npm install ccxt
```

```
var ccxt = require ('ccxt')

console.log (ccxt.exchanges) // print all available exchanges
```

3.1.1 Node.js + Windows

Windows users having difficulties installing `w3`, `scrypt` or `node-gyp` dependencies for the ccxt library, try installing `scrypt` first:

```
npm install -g web3 --unsafe-perm=true --allow-root
```

or

```
sudo npm install -g web3 --unsafe-perm=true --allow-root
```

Then install ccxt as usual with `npm install ccxt`.

If that does not help, please, follow here: https://github.com/nodejs/node-gyp#on-windows

3.2 JavaScript (for use with the `<script>` tag):

All-in-one browser bundle (dependencies included), served from unpkg CDN, which is a fast, global content delivery network for everything on NPM.

```
<script type="text/javascript" src="https://unpkg.com/ccxt"></script>
```

Creates a global `ccxt` object:

```
console.log (ccxt.exchanges) // print all available exchanges
```

3.3 Python

ccxt algotrading library in PyPI

```
pip install ccxt
```

```
import ccxt
print(ccxt.exchanges) # print a list of all available exchange classes
```

The library supports concurrent asynchronous mode with asyncio and async/await in Python 3.5.3+

```
import ccxt.async_support as ccxt # link against the asynchronous version of ccxt
```

3.4 PHP

The autoloadable version of ccxt can be installed with Packagist/Composer (PHP 5.3+).

It can also be installed from the source code: `ccxt.php <https://raw.githubusercontent.com/ccxt/ccxt/master/php>`__

It requires common PHP modules:

- cURL
- mbstring (using UTF-8 is highly recommended)

- PCRE
- iconv
- gmp (this is a built-in extension as of PHP 7.2+)

```
include "ccxt.php";
var_dump (\ccxt\Exchange::$exchanges); // print a list of all available exchange␣
↪classes
```

CHAPTER 4

Proxy

In some specific cases you may want a proxy, if you experience issues with DDoS protection by Cloudflare or your network / country / IP is rejected by their filters.

Bear in mind that each added intermediary contributes to the overall latency and roundtrip time. Longer delays can result in price slippage.

4.1 JavaScript Proxies

In order to use proxies with JavaScript, one needs to pass the proxying `agent` option to the exchange class instance constructor (or set the `exchange.agent` property later after instantiation in runtime):

```
const ccxt = require ('ccxt')
    , HttpsProxyAgent = require ('https-proxy-agent')

const proxy = process.env.http_proxy || 'http://168.63.76.32:3128' // HTTP/HTTPS␣
↪proxy to connect to
const agent = new HttpsProxyAgent (proxy)

const kraken = new ccxt.kraken ({ agent })
```

4.2 Python Proxies

The python version of the library uses the python-requests package for underlying HTTP and supports all means of customization available in the `requests` package, including proxies.

You can configure proxies by setting the environment variables HTTP_PROXY and HTTPS_PROXY.

```
$ export HTTP_PROXY="http://10.10.1.10:3128"
$ export HTTPS_PROXY="http://10.10.1.10:1080"
```

After exporting the above variables with your proxy settings, all reqeusts from within ccxt will be routed through those proxies.

You can also set them programmatically:

```
import ccxt
exchange = ccxt.poloniex({
    'proxies': {
        'http': 'http://10.10.1.10:3128',  # these proxies won't work for you, they␣
↪are here for example
        'https': 'https://10.10.1.10:1080',
    },
})
```

Or

```
import ccxt
exchange = ccxt.poloniex()
exchange.proxies = {
  'http': 'http://10.10.1.10:3128', # these proxies won't work for you, they are here␣
↪for example
  'https': 'https://10.10.1.10:1080',
}
```

4.2.1 Python 2 and 3 sync proxies

- https://github.com/ccxt/ccxt/blob/master/examples/py/proxy-sync-python-requests-2-and-3.py

```
# -*- coding: utf-8 -*-

import os
import sys
import ccxt
from pprint import pprint


exchange = ccxt.poloniex({
    #
    # ↓ The "proxy" property setting below is for CORS-proxying only!
    # Do not use it if you don't know what a CORS proxy is.
    # https://github.com/ccxt/ccxt/wiki/Install#cors-access-control-allow-origin
    # You should only use the "proxy" setting if you're having a problem with Access-
↪Control-Allow-Origin
    # In Python you rarely need to use it, if ever at all.
    #
    # 'proxy': 'https://cors-anywhere.herokuapp.com/',
    #
    # ↓ On the other hand, the "proxies" setting is for HTTP(S)-proxying (SOCKS, etc...
↪)
    # It is a standard method of sending your requests through your proxies
    # This gets passed to the `python-requests` implementation directly
    # You can also enable this with environment variables, as described here:
    # http://docs.python-requests.org/en/master/user/advanced/#proxies
    # This is the setting you should be using with synchronous version of ccxt in␣
↪Python 2 and 3
    #
    'proxies': {
```

(continues on next page)

(continued from previous page)

```
        'http': 'http://10.10.1.10:3128',
        'https': 'http://10.10.1.10:1080',
    },
})

# your code goes here...

pprint(exchange.fetch_ticker('ETH/BTC'))
```

4.2.2 Python 3.5+ asyncio/aiohttp proxy

- https://github.com/ccxt/ccxt/blob/master/examples/py/proxy-asyncio-aiohttp-python-3.py

```
# -*- coding: utf-8 -*-

import asyncio
import os
import sys
import ccxt.async_support as ccxt
from pprint import pprint


async def test_gdax():

    exchange = ccxt.poloniex({
        #
        # ↓ The "proxy" property setting below is for CORS-proxying only!
        # Do not use it if you don't know what a CORS proxy is.
        # https://github.com/ccxt/ccxt/wiki/Install#cors-access-control-allow-origin
        # You should only use the "proxy" setting if you're having a problem with
↪Access-Control-Allow-Origin
        # In Python you rarely need to use it, if ever at all.
        #
        # 'proxy': 'https://cors-anywhere.herokuapp.com/',
        #
        # ↓ The "aiohttp_proxy" setting is for HTTP(S)-proxying (SOCKS, etc...)
        # It is a standard method of sending your requests through your proxies
        # This gets passed to the `asyncio` and `aiohttp` implementation directly
        # You can use this setting as documented here:
        # https://docs.aiohttp.org/en/stable/client_advanced.html#proxy-support
        # This is the setting you should be using with async version of ccxt in
↪Python 3.5+
        #
        'aiohttp_proxy': 'http://proxy.com',
        # 'aiohttp_proxy': 'http://user:pass@some.proxy.com',
        # 'aiohttp_proxy': 'http://10.10.1.10:3128',
    })

    # your code goes here...

    ticker = await exchange.fetch_ticker('ETH/BTC')

    # don't forget to free the used resources, when you don't need them anymore
    await exchange.close()
```

(continues on next page)

(continued from previous page)

```
    return ticker

if __name__ == '__main__':
    pprint(asyncio.get_event_loop().run_until_complete(test_gdax()))
```

A more detailed documentation on using proxies with the sync python version of the ccxt library can be found here:

- Proxies
- SOCKS

CHAPTER 5

CORS (Access-Control-Allow-Origin)

If you need a CORS proxy, use the `proxy` property (a string literal) containing base URL of http(s) proxy. It is for use with web browsers and from blocked locations.

CORS is Cross-Origin Resource Sharing. When accessing the HTTP REST API of an exchange from browser with ccxt library you may get a warning or an exception, saying `No 'Access-Control-Allow-Origin' header is present on the requested resource`. That means that the exchange admins haven't enabled access to their API from arbitrary web browser pages.

You can still use the ccxt library from your browser via a CORS-proxy, which is very easy to set up or install. There are also public CORS proxies on the internet.

The absolute exchange endpoint URL is appended to `proxy` string before HTTP request is sent to exchange. The `proxy` setting is an empty string `''` by default. Below are examples of a non-empty `proxy` string (last slash is mandatory!):

- `kraken.proxy = 'https://crossorigin.me/'`
- `gdax.proxy = 'https://cors-anywhere.herokuapp.com/'`

To run your own CORS proxy locally you can either set up one of the existing ones or make a quick script of your own, like shown below.

5.1 Node.js CORS Proxy

```
// JavaScript CORS Proxy
// Save this in a file like cors.js and run with `node cors [port]`
// It will listen for your requests on the port you pass in command line or port 8080␣
↪by default
let port = (process.argv.length > 2) ? parseInt (process.argv[2]) : 8080; // default
require ('cors-anywhere').createServer ().listen (port, 'localhost')
```

5.2 Python CORS Proxy

```
#!/usr/bin/env python
# Python CORS Proxy
# Save this in a file like cors.py and run with `python cors.py [port]` or `cors␣
↪[port]`
try:
    # Python 3
    from http.server import HTTPServer, SimpleHTTPRequestHandler, test as test_orig
    import sys
    def test (*args):
        test_orig (*args, port = int (sys.argv[1]) if len (sys.argv) > 1 else 8080)
except ImportError: # Python 2
    from BaseHTTPServer import HTTPServer, test
    from SimpleHTTPServer import SimpleHTTPRequestHandler

class CORSRequestHandler (SimpleHTTPRequestHandler):
    def end_headers (self):
        self.send_header ('Access-Control-Allow-Origin', '*')
        SimpleHTTPRequestHandler.end_headers (self)

if __name__ == '__main__':
    test (CORSRequestHandler, HTTPServer)
```

5.3 Testing CORS

After you set it up and run it, you can test it by querying the target URL of exchange endpoint through the proxy (like https://localhost:8080/https://exchange.com/path/to/endpoint).

To test the CORS you can do either of the following:

- set up proxy somewhere in your browser settings, then go to endpoint URL `https://exchange.com/path/to/endpoint`
- type that URL directly in the address bar as `https://localhost:8080/https://exchange.com/path/to/endpoint`
- cURL it from command like `curl https://localhost:8080/https://exchange.com/path/to/endpoint`

To let ccxt know of the proxy, you can set the `proxy` property on your exchange instance.

CHAPTER 6

Overview

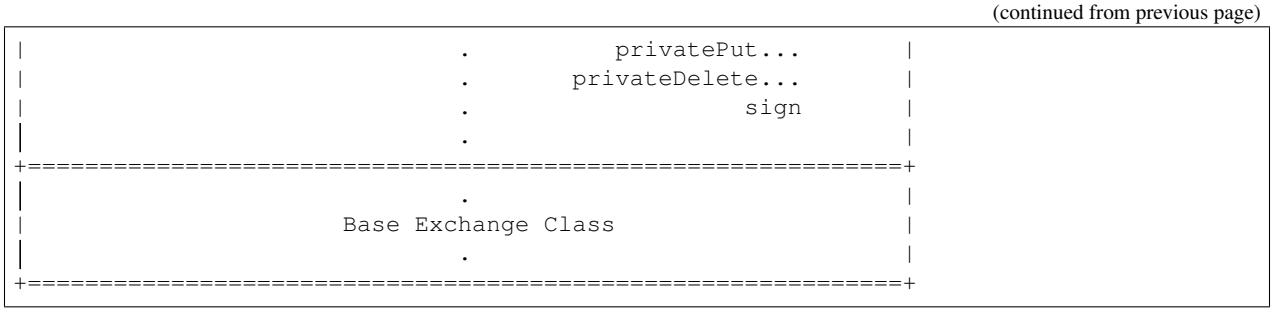
The ccxt library is a collection of available crypto *exchanges* or exchange classes. Each class implements the public and private API for a particular crypto exchange. All exchanges are derived from the base Exchange class and share a set of common methods. To access a particular exchange from ccxt library you need to create an instance of corresponding exchange class. Supported exchanges are updated frequently and new exchanges are added regularly.

The structure of the library can be outlined as follows:

```
                            User
+-------------------------------------------------------------+
|                           CCXT                              |
+-----------------------------+-----------------------------+
|            Public           |           Private            |
+=============================================================+
|                             .                              |
|                    The Unified CCXT API                    |
|                             .                              |
|       loadMarkets           .           fetchBalance       |
|       fetchMarkets          .            createOrder       |
|       fetchCurrencies       .            cancelOrder       |
|       fetchTicker           .             fetchOrder       |
|       fetchTickers          .            fetchOrders       |
|       fetchOrderBook        .        fetchOpenOrders       |
|       fetchOHLCV            .      fetchClosedOrders       |
|       fetchTrades           .          fetchMyTrades       |
|                             .                deposit       |
|                             .               withdraw       |
|                             .                              |
+=============================================================+
|                             .                              |
|                     Custom Exchange API                    |
|         (Derived Classes And Their Implicit Methods)       |
|                             .                              |
|       publicGet...          .          privateGet...       |
|       publicPost...         .         privatePost...       |
```

(continues on next page)

(continued from previous page)



Full public and private HTTP REST APIs for all exchanges are implemented. WebSocket and FIX implementations in JavaScript, PHP, Python and other languages coming soon.

- *Exchanges*
- *Markets*
- *API Methods / Endpoints*
- *Market Data*
- *Trading*

CHAPTER 7

Exchanges

The ccxt library currently supports the following 131 cryptocurrency exchange markets and trading APIs:

	id	name	certified	ver	doc	countries
	_1broker	1Broker		2	API	US
	_1btcxe	1BTCXE		*	API	Panama
	acx	ACX		2	API	Australia
	allcoin	Allcoin		1	API	Canada
	anxpro	ANXPro		2	API	Japan, Singapore, Hong Kon
	anybits	Anybits		*	API	Ireland
	bcex	BCEX		1	API	China, Canada
	bibox	Bibox		1	API	China, US, South Korea
	bigone	BigONE		2	API	UK
	binance	Binance	CCXT Certified	*	API	Japan
	bit2c	Bit2C		*	API	Israel

Continue

Table 1 – continued from previous page

	id	name	certified	ver	doc	countries
	bitbank	bitbank		1	API	Japan
	bitbay	BitBay		*	API	Malta, EU
	bitfinex	Bitfinex	CCXT Certified	1	API	British Virgin Islands
	bitfinex2	Bitfinex v2		2	API	British Virgin Islands
	bitflyer	bitFlyer		1	API	Japan
	bitforex	Bitforex		1	API	China
	bithumb	Bithumb		*	API	South Korea
	bitkk	bitkk		1	API	China
	bitlish	Bitlish		1	API	UK, EU, Russia
	bitmarket	BitMarket		*	API	Poland, EU
	bitmex	BitMEX		1	API	Seychelles
	bitsane	Bitsane		*	API	Ireland
	bitso	Bitso		3	API	Mexico
	bitstamp	Bitstamp		2	API	UK
	bitstamp1	Bitstamp v1		1	API	UK
	bittrex	Bittrex	CCXT Certified	1.1	API	US
	bitz	Bit-Z		2	API	Hong Kong
	bl3p	BL3P		1	API	Netherlands, EU
	bleutrade	Bleutrade		2	API	Brazil
	braziliex	Braziliex		*	API	Brazil
	btcalpha	BTC-Alpha		1	API	US

Continued

Table 1 – continued from previous page

	id	name	certified	ver	doc	countries
	btcbox	BtcBox		1	API	Japan
	btcchina	BTCChina		1	API	China
	btcexchange	BTCExchange		*	API	Philippines
	btcmarkets	BTC Markets		*	API	Australia
	btctradeim	BtcTrade.im		*	API	Hong Kong
	btctradeua	BTC Trade UA		*	API	Ukraine
	btcturk	BTCTurk		*	API	Turkey
	btcx	BTCX		1	API	Iceland, US, EU
	bxinth	BX.in.th		*	API	Thailand
	ccex	C-CEX		*	API	Germany, EU
	cex	CEX.IO		*	API	UK, EU, Cyprus, Russia
	chbtc	CHBTC		1	API	China
	chilebit	ChileBit		1	API	Chile
	cobinhood	COBINHOOD		1	API	Taiwan
	coinbase	coinbase		2	API	US
	coinbaseprime	Coinbase Prime		*	API	US
	coinbasepro	Coinbase Pro		*	API	US
	coincheck	coincheck		*	API	Japan, Indonesia
	coinegg	CoinEgg		*	API	China, UK
	coinex	CoinEx		1	API	China
	coinexchange	CoinExchange		*	API	India, Japan, South Korea, V

Continue

Table 1 – continued from previous page

	id	name	certified	ver	doc	countries
	coinfalcon	CoinFalcon		1	API	UK
	coinfloor	coinfloor		*	API	UK
	coingi	Coingi		*	API	Panama, Bulgaria, China, US
	coinmarketcap	CoinMarketCap		1	API	US
	coinmate	CoinMate		*	API	UK, Czech Republic, EU
	coinnest	coinnest		*	API	South Korea
	coinone	CoinOne		2	API	South Korea
	coinsecure	Coinsecure		1	API	India
	coinspot	CoinSpot		*	API	Australia
	cointiger	CoinTiger		1	API	China
	coolcoin	CoolCoin		*	API	Hong Kong
	crypton	Crypton		1	API	EU
	cryptopia	Cryptopia		*	API	New Zealand
	deribit	Deribit		1	API	Netherlands
	dsx	DSX		3	API	UK
	ethfinex	Ethfinex		1	API	British Virgin Islands
	exmo	EXMO		1	API	Spain, Russia
	exx	EXX		*	API	China
	fcoin	FCoin		2	API	China
	flowbtc	flowBTC		1	API	Brazil
	foxbit	FoxBit		1	API	Brazil

Continued

Table 1 – continued from previous page

	id	name	certified	ver	doc	countries
	fybse	FYB-SE		*	API	Sweden
	fybsg	FYB-SG		*	API	Singapore
	gatecoin	Gatecoin		*	API	Hong Kong
	gateio	Gate.io		2	API	China
	gdax	GDAX		*	API	US
	gemini	Gemini		1	API	US
	getbtc	GetBTC		*	API	St. Vincent & Grenadines, R
	hadax	HADAX		1	API	China
	hitbtc	HitBTC		1	API	Hong Kong
	hitbtc2	HitBTC v2		2	API	Hong Kong
	huobi	Huobi		3	API	China
	huobicny	Huobi CNY		1	API	China
	huobipro	Huobi Pro		1	API	China
	ice3x	ICE3X		1	API	South Africa
	independentreserve	Independent Reserve		*	API	Australia, New Zealand
	indodax	INDODAX		1.8	API	Indonesia
	itbit	itBit		1	API	US
	jubi	jubi.com		1	API	China
	kraken	Kraken	CCXT Certified	0	API	US
	kucoin	Kucoin		1	API	Hong Kong
	kuna	Kuna		2	API	Ukraine

Continue

Table 1 – continued from previous page

	id	name	certified	ver	doc	countries
	lakebtc	LakeBTC		2	API	US
	lbank	LBank		1	API	China
	liqui	Liqui		3	API	Ukraine
	livecoin	LiveCoin		*	API	US, UK, Russia
	luno	luno		1	API	UK, Singapore, South Africa
	lykke	Lykke		1	API	Switzerland
	mercado	Mercado Bitcoin		3	API	Brazil
	mixcoins	MixCoins		1	API	UK, Hong Kong
	negociecoins	NegocieCoins		3	API	Brazil
	nova	Novaexchange		2	API	Tanzania
	okcoincny	OKCoin CNY		1	API	China
	okcoinusd	OKCoin USD		1	API	China, US
	okex	OKEX		1	API	China, US
	paymium	Paymium		1	API	France, EU
	poloniex	Poloniex		*	API	US
	qryptos	QRYPTOS		2	API	China, Taiwan
	quadrigacx	QuadrigaCX		2	API	Canada
	quoinex	QUOINEX		2	API	Japan, Singapore, Vietnam
	rightbtc	RightBTC		*	API	United Arab Emirates
	southxchange	SouthXchange		*	API	Argentina
	surbitcoin	SurBitcoin		1	API	Venezuela

Continued

Table 1 – continued from previous page

	id	name	certified	ver	doc	countries
The Ocean	theocean	The Ocean	CCXT Certified	0	API	US
THEROCK TRADING LTD.	therock	TheRockTrading		1	API	Malta
TideBit	tidebit	TideBit		2	API	Hong Kong
TIDEX	tidex	Tidex		3	API	UK
UEX	uex	UEX		1.0.3	API	Singapore, US
URDUBIT	urdubit	UrduBit		1	API	Pakistan
Vaultoro	vaultoro	Vaultoro		1	API	Switzerland
vbtc	vbtc	VBTC		1	API	Vietnam
VirWoX	virwox	VirWoX		*	API	Austria, EU
WEX	wex	WEX		3	API	New Zealand
xBTCe Bitcoin Exchange	xbtce	xBTCe		1	API	Russia
YObit.net	yobit	YoBit		3	API	Russia
云币	yunbi	YUNBI		2	API	China
Zaif	zaif	Zaif		1	API	Japan
ZB.COM	zb	ZB		1	API	China

Besides making basic market and limit orders, some exchanges offer margin trading (leverage), various derivatives (like futures contracts and options) and also have dark pools, OTC (over-the-counter trading), merchant APIs and much more.

7.1 Instantiation

To connect to an exchange and start trading you need to instantiate an exchange class from ccxt library.

To get the full list of ids of supported exchanges programmatically:

```
// JavaScript
const ccxt = require ('ccxt')
console.log (ccxt.exchanges)
```

```python
# Python
import ccxt
print (ccxt.exchanges)
```

```php
// PHP
include 'ccxt.php';
var_dump (\ccxt\Exchange::$exchanges);
```

An exchange can be instantiated like shown in the examples below:

```javascript
// JavaScript
const ccxt = require ('ccxt')
let exchange = new ccxt.kraken () // default id
let kraken1 = new ccxt.kraken ({ id: 'kraken1' })
let kraken2 = new ccxt.kraken ({ id: 'kraken2' })
let id = 'gdax'
let gdax = new ccxt[id] ();

// from variable id
const exchangeId = 'binance'
    , exchangeClass = ccxt[exchangeId]
    , exchange = new exchangeClass ({
        'apiKey': 'YOUR_API_KEY',
        'secret': 'YOUR_SECRET',
        'timeout': 30000,
        'enableRateLimit': true,
    })
```

```python
# Python
import ccxt
exchange = ccxt.okcoinusd () # default id
okcoin1 = ccxt.okcoinusd ({ 'id': 'okcoin1' })
okcoin2 = ccxt.okcoinusd ({ 'id': 'okcoin2' })
id = 'btcchina'
btcchina = eval ('ccxt.%s ()' % id)
gdax = getattr (ccxt, 'gdax') ()

# from variable id
exchange_id = 'binance'
exchange_class = getattr(ccxt, exchange_id)
exchange = exchange_class({
    'apiKey': 'YOUR_API_KEY',
    'secret': 'YOUR_SECRET',
    'timeout': 30000,
    'enableRateLimit': True,
})
```

The ccxt library in PHP uses builtin UTC/GMT time functions, therefore you are required to set date.timezone in your php.ini or call date_default_timezone_set () function before using the PHP version of the library. The recommended timezone setting is "UTC".

```php
// PHP
date_default_timezone_set ('UTC');
include 'ccxt.php';
$bitfinex = new \ccxt\bitfinex (); // default id
$bitfinex1 = new \ccxt\bitfinex (array ('id' => 'bitfinex1'));
```

(continues on next page)

(continued from previous page)

```
$bitfinex2 = new \ccxt\bitfinex (array ('id' => 'bitfinex2'));
$id = 'kraken';
$exchange = '\\ccxt\\' . $id
$kraken = new $exchange ();

// from variable id
$exchange_id = 'binance';
$exchange_class = "\\ccxt\\$exchange_id";
$exchange = new $exchange_class (array (
    'apiKey' => 'YOUR_API_KEY',
    'secret' => 'YOUR_SECRET',
    'timeout' => 30000,
    'enableRateLimit' => true,
));
```

7.2 Exchange Structure

Every exchange has a set of properties and methods, most of which you can override by passing an associative array of params to an exchange constructor. You can also make a subclass and override everything.

Here's an overview of base exchange properties with values added for example:

```
{
    'id':   'exchange'                  // lowercase string exchange id
    'name': 'Exchange'                  // human-readable string
    'countries': [ 'US', 'CN', 'EU' ],  // array of ISO country codes
    'urls': {
        'api': 'https://api.example.com/data',  // string or dictionary of base API␣
↪URLs
        'www': 'https://www.example.com'        // string website URL
        'doc': 'https://docs.example.com/api',  // string URL or array of URLs
    },
    'version':         'v1',            // string ending with digits
    'api':             { ... },         // dictionary of api endpoints
    'has': {                            // exchange capabilities
        'CORS': false,
        'publicAPI': true,
        'privateAPI': true,
        'cancelOrder': true,
        'createDepositAddress': false,
        'createOrder': true,
        'deposit': false,
        'fetchBalance': true,
        'fetchClosedOrders': false,
        'fetchCurrencies': false,
        'fetchDepositAddress': false,
        'fetchMarkets': true,
        'fetchMyTrades': false,
        'fetchOHLCV': false,
        'fetchOpenOrders': false,
        'fetchOrder': false,
        'fetchOrderBook': true,
        'fetchOrders': false,
        'fetchTicker': true,
```

(continues on next page)

(continued from previous page)

```
        'fetchTickers': false,
        'fetchBidsAsks': false,
        'fetchTrades': true,
        'withdraw': false,
    },
    'timeframes': {                     // empty if the exchange !has.fetchOHLCV
        '1m': '1minute',
        '1h': '1hour',
        '1d': '1day',
        '1M': '1month',
        '1y': '1year',
    },
    'timeout':          10000,          // number in milliseconds
    'rateLimit':        2000,           // number in milliseconds
    'userAgent':       'ccxt/1.1.1 ...' // string, HTTP User-Agent header
    'verbose':          false,          // boolean, output error details
    'markets':         { ... }          // dictionary of markets/pairs by symbol
    'symbols':         [ ... ]          // sorted list of string symbols (traded␣
↪pairs)
    'currencies':      { ... }          // dictionary of currencies by currency code
    'markets_by_id':   { ... },         // dictionary of dictionaries (markets) by id
    'proxy': 'https://crossorigin.me/', // string URL
    'apiKey':   '92560ffae9b8a0421...', // string public apiKey (ASCII, hex, Base64, .
↪..)
    'secret':   '9aHjPmW+EtRRKN/Oi...'  // string private secret key
    'password': '6kszf4aci8r',          // string password
    'uid':      '123456',               // string user id
}
```

7.2.1 Exchange Properties

Below is a detailed description of each of the base exchange properties:

- `id`: Each exchange has a default id. The id is not used for anything, it's a string literal for user-land exchange instance identification purposes. You can have multiple links to the same exchange and differentiate them by ids. Default ids are all lowercase and correspond to exchange names.
- `name`: This is a string literal containing the human-readable exchange name.
- `countries`: An array of string literals of 2-symbol ISO country codes, where the exchange is operating from.
- `urls['api']`: The single string literal base URL for API calls or an associative array of separate URLs for private and public APIs.
- `urls['www']`: The main HTTP website URL.
- `urls['doc']`: A single string URL link to original documentation for exchange API on their website or an array of links to docs.
- `version`: A string literal containing version identifier for current exchange API. The ccxt library will append this version string to the API Base URL upon each request. You don't have to modify it, unless you are implementing a new exchange API. The version identifier is a usually a numeric string starting with a letter 'v' in some cases, like v1.1. Do not override it unless you are implementing your own new crypto exchange class.
- `api`: An associative array containing a definition of all API endpoints exposed by a crypto exchange. The API definition is used by ccxt to automatically construct callable instance methods for each available endpoint.
- `has`: This is an associative array of exchange capabilities (e.g `fetchTickers`, `fetchOHLCV` or `CORS`).

- `timeframes`: An associative array of timeframes, supported by the fetchOHLCV method of the exchange. This is only populated when `has['fetchOHLCV']` property is true.
- `timeout`: A timeout in milliseconds for a request-response roundtrip (default timeout is 10000 ms = 10 seconds). You should always set it to a reasonable value, hanging forever with no timeout is not your option, for sure.
- `rateLimit`: A request rate limit in milliseconds. Specifies the required minimal delay between two consequent HTTP requests to the same exchange. The built-in rate-limiter is disabled by default and is turned on by setting the `enableRateLimit` property to true.
- `enableRateLimit`: A boolean (true/false) value that enables the built-in rate limiter and throttles consecutive requests. This setting is false (disabled) by default. **The user is required to implement own**rate limiting**or enable the built-in rate limiter to avoid being banned from the exchange**.
- `userAgent`: An object to set HTTP User-Agent header to. The ccxt library will set its User-Agent by default. Some exchanges may not like it. If you are having difficulties getting a reply from an exchange and want to turn User-Agent off or use the default one, set this value to false, undefined, or an empty string.
- `verbose`: A boolean flag indicating whether to log HTTP requests to stdout (verbose flag is false by default). Python people have an alternative way of DEBUG logging with a standard pythonic logger, which is enabled by adding these two lines to the beginning of their code:

```
import logging
logging.basicConfig(level=logging.DEBUG)
```

- `markets`: An associative array of markets indexed by common trading pairs or symbols. Markets should be loaded prior to accessing this property. Markets are unavailable until you call the `loadMarkets()` / `load_markets()` method on exchange instance.
- `symbols`: A non-associative array (a list) of symbols available with an exchange, sorted in alphabetical order. These are the keys of the `markets` property. Symbols are loaded and reloaded from markets. This property is a convenient shorthand for all market keys.
- `currencies`: An associative array (a dict) of currencies by codes (usually 3 or 4 letters) available with an exchange. Currencies are loaded and reloaded from markets.
- `markets_by_id`: An associative array of markets indexed by exchange-specific ids. Markets should be loaded prior to accessing this property.
- `proxy`: A string literal containing base URL of http(s) proxy, `''` by default. For use with web browsers and from blocked locations. An example of a proxy string is `'http://crossorigin.me/'`. The absolute exchange endpoint URL is appended to this string before sending the HTTP request.
- `apiKey`: This is your public API key string literal. Most exchanges require this for trading (see below).
- `secret`: Your private secret API key string literal. Most exchanges require this as well together with the apiKey.
- `password`: A string literal with your password/phrase. Some exchanges require this parameter for trading, but most of them don't.
- `uid`: A unique id of your account. This can be a string literal or a number. Some exchanges also require this for trading, but most of them don't.
- `has`: An assoc-array containing flags for exchange capabilities, including the following:

```
'has': {

    'CORS': false,  // has Cross-Origin Resource Sharing enabled (works from␣
↪browser) or not
```

(continues on next page)

(continued from previous page)

```
    'publicAPI': true,  // has public API available and implemented, true/false
    'privateAPI': true, // has private API available and implemented, true/false

    // unified methods availability flags (can be true, false, or 'emulated'):

    'cancelOrder': true,
    'createDepositAddress': false,
    'createOrder': true,
    'deposit': false,
    'fetchBalance': true,
    'fetchClosedOrders': false,
    'fetchCurrencies': false,
    'fetchDepositAddress': false,
    'fetchMarkets': true,
    'fetchMyTrades': false,
    'fetchOHLCV': false,
    'fetchOpenOrders': false,
    'fetchOrder': false,
    'fetchOrderBook': true,
    'fetchOrders': false,
    'fetchTicker': true,
    'fetchTickers': false,
    'fetchBidsAsks': false,
    'fetchTrades': true,
    'withdraw': false,
}
```

The meaning of each flag showing availability of this or that method is:

- boolean `true` means the method is natively available from the exchange API and unified in the ccxt library
- boolean `false` means the method isn't natively available from the exchange API or not unified in the ccxt library yet
- an `'emulated'` string means the endpoint isn't natively available from the exchange API but reconstructed by the ccxt library from available true-methods

7.3 Rate Limit

Exchanges usually impose what is called a *rate limit*. Exchanges will remember and track your user credentials and your IP address and will not allow you to query the API too frequently. They balance their load and control traffic congestion to protect API servers from (D)DoS and misuse.

WARNING: Stay under the rate limit to avoid ban!

Most exchanges allow **up to 1 or 2 requests per second**. Exchanges may temporarily restrict your access to their API or ban you for some period of time if you are too aggressive with your requests.

The ‘‘exchange.rateLimit‘‘ property is set to a safe default which is sub-optimal. Some exchanges may have varying rate limits for different endpoints. It is up to the user to tweak ‘‘rateLimit‘‘ according to application-specific purposes.

The CCXT library has a built-in experimental rate-limiter that will do the necessary throttling in background transparently to the user. **WARNING: users are responsible for at least some type of rate-limiting: either by implementing a custom algorithm or by doing it with the built-in rate-limiter.**.

Turn on the built-in rate-limiter with `.enableRateLimit` property, like so:

```
// JavaScript

// enable built-in rate limiting upon instantiation of the exchange
const exchange = new ccxt.bitfinex ({
    'enableRateLimit': true,
})

// or switch the built-in rate-limiter on or off later after instantiation
exchange.enableRateLimit = true // enable
exchange.enableRateLimit = false // disable
```

```
# Python

# enable built-in rate limiting upon instantiation of the exchange
exchange = ccxt.bitfinex({
    'enableRateLimit': True,
})

# or switch the built-in rate-limiter on or off later after instantiation
exchange.enableRateLimit = True  # enable
exchange.enableRateLimit = False  # disable
```

```
// PHP

// enable built-in rate limiting upon instantiation of the exchange
$exchange = new \ccxt\bitfinex (array (
    'enableRateLimit' => true,
));

// or switch the built-in rate-limiter on or off later after instantiation
$exchange->enableRateLimit = true; // enable
$exchange->enableRateLimit = false; // disable
```

In case your calls hit a rate limit or get nonce errors, the ccxt library will throw an exception of one of the following types:

- DDoSProtectionError
- ExchangeNotAvailable
- ExchangeError

A later retry is usually enough to handle that. More on that here:

- Authentication
- Troubleshooting
- Overriding The Nonce

7.3.1 DDoS Protection By Cloudflare / Incapsula

Some exchanges are DDoS-protected by Cloudflare or Incapsula. Your IP can get temporarily blocked during periods of high load. Sometimes they even restrict whole countries and regions. In that case their servers usually return a page that states a HTTP 40x error or runs an AJAX test of your browser / captcha test and delays the reload of the page for several seconds. Then your browser/fingerprint is granted access temporarily and gets added to a whitelist or receives a HTTP cookie for further use.

The most common symptoms for a DDoS protection problem, rate-limiting problem or for a location-based filtering issue: - Getting `RequestTimeout` exceptions with all types of exchange methods - Catching `ExchangeError` or `ExchangeNotAvailable` with HTTP error codes 400, 403, 404, 429, 500, 501, 503, etc.. - Having DNS resolving issues, SSL certificate issues and low-level connectivity issues - Getting a template HTML page instead of JSON from the exchange

If you encounter DDoS protection errors and cannot reach a particular exchange then:

- try using a cloudscraper:
  - https://github.com/ccxt/ccxt/blob/master/examples/js/bypass-cloudflare.js
  - https://github.com/ccxt/ccxt/blob/master/examples/py/bypass-cloudflare.py
  - https://github.com/ccxt/ccxt/blob/master/examples/py/bypass-cloudflare-with-cookies.py
- use a proxy (this is less responsive, though)
- ask the exchange support to add you to a whitelist
- run your software in close proximity to the exchange (same country, same city, same datacenter, same server rack, same server)
- try an alternative IP within a different geographic region
- run your software in a distributed network of servers
- ...

CHAPTER 8

Markets

Each exchange is a place for trading some kinds of valuables. Sometimes they are called with various different terms like instruments, symbols, trading pairs, currencies, tokens, stocks, commodities, contracts, etc, but they all mean the same – a trading pair, a symbol or a financial instrument.

In terms of the ccxt library, every exchange offers multiple markets within itself. The set of markets differs from exchange to exchange opening possibilities for cross-exchange and cross-market arbitrage. A market is usually a pair of traded crypto/fiat currencies.

8.1 Market Structure

```
{
    'id':     'btcusd',    // string literal for referencing within an exchange
    'symbol': 'BTC/USD',   // uppercase string literal of a pair of currencies
    'base':   'BTC',       // uppercase string, base currency, 3 or more letters
    'quote':  'USD',       // uppercase string, quote currency, 3 or more letters
    'active': true,        // boolean, market status
    'precision': {         // number of decimal digits "after the dot"
        'price': 8,        // integer
        'amount': 8,       // integer
        'cost': 8,         // integer
    },
    'limits': {            // value limits when placing orders on this market
        'amount': {
            'min': 0.01,   // order amount should be > min
            'max': 1000,   // order amount should be < max
        },
        'price': { ... }, // same min/max limits for the price of the order
        'cost':  { ... }, // same limits for order cost = price * amount
    },
    'info':      { ... }, // the original unparsed market info from the exchange
}
```

Each market is an associative array (aka dictionary) with the following keys:

- `id`. The string or numeric ID of the market or trade instrument within the exchange. Market ids are used inside exchanges internally to identify trading pairs during the request/response process.
- `symbol`. An uppercase string code representation of a particular trading pair or instrument. This is usually written as `BaseCurrency/QuoteCurrency` with a slash as in `BTC/USD`, `LTC/CNY` or `ETH/EUR`, etc. Symbols are used to reference markets within the ccxt library (explained below).
- `base`. An uppercase string code of base fiat or crypto currency.
- `quote`. An uppercase string code of quoted fiat or crypto currency.
- `active`. A boolean indicating whether or not trading this market is currently possible.
- `info`. An associative array of non-common market properties, including fees, rates, limits and other general market information. The internal info array is different for each particular market, its contents depend on the exchange.
- `precision`. The amounts of decimal digits accepted in order values by exchanges upon order placement for price, amount and cost.
- `limits`. The minimums and maximums for prices, amounts (volumes) and costs (where cost = price * amount).

8.1.1 Precision And Limits

Do not confuse ‘‘limits‘‘ with ‘‘precision‘‘! Precision has nothing to do with min limits. A precision of 8 digits does not necessarily mean a min limit of 0.00000001. The opposite is also true: a min limit of 0.0001 does not necessarily mean a precision of 4.

Examples:

1. `(market['limits']['amount']['min'] == 0.05) && (market['precision']['amount'] == 4)`

In the first example the **amount** of any order placed on the market **must satisfy both conditions**:

- The *amount value* should be >= 0.05:

```
+ good: 0.05, 0.051, 0.0501, 0.0502, ..., 0.0599, 0.06, 0.0601, ...
- bad: 0.04, 0.049, 0.0499
```

- *Precision of the amount* should up to 4 decimal digits:

```
+ good: 0.05, 0.051, 0.052, ..., 0.0531, ..., 0.06, ... 0.0719, ...
- bad: 0.05001, 0.05000, 0.06001
```

2. `(market['limits']['price']['min'] == 0.0019) && (market['precision']['price'] == 5)`

In the second example the **price** of any order placed on the market **must satisfy both conditions**:

- The *price value* should be >= 0.019:

```
+ good: 0.019, ... 0.0191, ... 0.01911, 0.01912, ...
- bad: 0.016, ..., 0.01699
```

- *Precision of price* should be 5 decimal digits or less:

```
+ good: 0.02, 0.021, 0.0212, 0.02123, 0.02124, 0.02125, ...
- bad: 0.017000, 0.017001, ...
```

3. `(market['limits']['amount']['min'] == 50) && (market['precision']['amount'] == -1)`

- The *amount value* should be greater than 50:

```
+ good: 50, 60, 70, 80, 90, 100, ... 2000, ...
- bad: 1, 2, 3, ..., 9
```

- A negative *amount precision* means that the amount should be an integer multiple of 10:

```
+ good: 50, ..., 110, ... 1230, ..., 1000000, ..., 1234560, ...
- bad: 9.5, ... 10.1, ..., 11, ... 200.71, ...
```

The ''precision'' and ''limits'' params are currently under heavy development, some of these fields may be missing here and there until the unification process is complete. This does not influence most of the orders but can be significant in extreme cases of very large or very small orders. The ''active'' flag is not yet supported and/or implemented by all markets.

8.2 Loading Markets

In most cases you are required to load the list of markets and trading symbols for a particular exchange prior to accessing other API methods. If you forget to load markets the ccxt library will do that automatically upon your first call to the unified API. It will send two HTTP requests, first for markets and then the second one for other data, sequentially.

In order to load markets manually beforehand call the `loadMarkets ()` / `load_markets ()` method on an exchange instance. It returns an associative array of markets indexed by trading symbol. If you want more control over the execution of your logic, preloading markets by hand is recommended.

```
// JavaScript
(async () => {
    let kraken = new ccxt.kraken ()
    let markets = await kraken.load_markets ()
    console.log (kraken.id, markets)
}) ()
```

```
# Python
okcoin = ccxt.okcoinusd ()
markets = okcoin.load_markets ()
print (okcoin.id, markets)
```

```
// PHP
$id = 'huobi';
$exchange = '\\ccxt\\' . $id;
$huobi = new $exchange ();
$markets = $huobi->load_markets ();
var_dump ($huobi->id, $markets);
```

8.3 Symbols And Market Ids

Market ids are used during the REST request-response process to reference trading pairs within exchanges. The set of market ids is unique per exchange and cannot be used across exchanges. For example, the BTC/USD pair/market may have different ids on various popular exchanges, like `btcusd`, `BTCUSD`, `XBTUSD`, `btc/usd`, `42` (numeric id),

`BTC/USD`, `Btc/Usd`, `tBTCUSD`, `XXBTZUSD`. You don't need to remember or use market ids, they are there for internal HTTP request-response purposes inside exchange implementations.

The ccxt library abstracts uncommon market ids to symbols, standardized to a common format. Symbols aren't the same as market ids. Every market is referenced by a corresponding symbol. Symbols are common across exchanges which makes them suitable for arbitrage and many other things.

A symbol is usually an uppercase string literal name for a pair of traded currencies with a slash in between. A currency is a code of three or four uppercase letters, like `BTC`, `ETH`, `USD`, `GBP`, `CNY`, `LTC`, `JPY`, `DOGE`, `RUB`, `ZEC`, `XRP`, `XMR`, etc. Some exchanges have exotic currencies with longer names. The first currency before the slash is usually called *base currency*, and the one after the slash is called *quote currency*. Examples of a symbol are: `BTC/USD`, `DOGE/LTC`, `ETH/EUR`, `DASH/XRP`, `BTC/CNY`, `ZEC/XMR`, `ETH/JPY`.

Sometimes the user might notice a symbol like `'XBTM18'` or `'.XRPUSDM20180101'` or some other *"exotic/rare symbols"*. The symbol is **not required** to have a slash or to be a pair of currencies. The string in the symbol really depends on the type of the market (whether it is a spot market or a futures market, a darkpool market or an expired market, etc). Attempting to parse the symbol string is highly discouraged, one should not rely on the symbol format, it is recommended to use market properties instead.

Market structures are indexed by symbols and ids. The base exchange class also has builtin methods for accessing markets by symbols. Most API methods require a symbol to be passed in their first argument. You are often required to specify a symbol when querying current prices, making orders, etc.

Most of the time users will be working with market symbols. You will get a standard userland exception if you access non-existent keys in these dicts.

```
// JavaScript

(async () => {

    console.log (await exchange.loadMarkets ())

    let btcusd1 = exchange.markets['BTC/USD']     // get market structure by symbol
    let btcusd2 = exchange.market ('BTC/USD')     // same result in a slightly␣
↪different way

    let btcusdId = exchange.marketId ('BTC/USD')  // get market id by symbol

    let symbols = exchange.symbols                // get an array of symbols
    let symbols2 = Object.keys (exchange.markets) // same as previous line

    console.log (exchange.id, symbols)            // print all symbols

    let currencies = exchange.currencies          // a list of currencies

    let bitfinex = new ccxt.bitfinex ()
    await bitfinex.loadMarkets ()

    bitfinex.markets['BTC/USD']                   // symbol → market (get market by␣
↪symbol)
    bitfinex.markets_by_id['XRPBTC']              // id → market (get market by id)

    bitfinex.markets['BTC/USD']['id']             // symbol → id (get id by symbol)
    bitfinex.markets_by_id['XRPBTC']['symbol']    // id → symbol (get symbol by id)

})
```

```python
# Python

print (exchange.load_markets ())

etheur1 = exchange.markets['ETH/EUR']       # get market structure by symbol
etheur2 = exchange.market ('ETH/EUR')       # same result in a slightly different way

etheurId = exchange.market_id ('BTC/USD')   # get market id by symbol

symbols = exchange.symbols                  # get a list of symbols
symbols2 = list (exchange.markets.keys ()) # same as previous line

print (exchange.id, symbols)                # print all symbols

currencies = exchange.currencies            # a list of currencies

kraken = ccxt.kraken ()
kraken.load_markets ()

kraken.markets['BTC/USD']                   # symbol → market (get market by symbol)
kraken.markets_by_id['XXRPZUSD']            # id → market (get market by id)

kraken.markets['BTC/USD']['id']             # symbol → id (get id by symbol)
kraken.markets_by_id['XXRPZUSD']['symbol'] # id → symbol (get symbol by id)
```

```php
// PHP

$var_dump ($exchange->load_markets ());

$dashcny1 = $exchange->markets['DASH/CNY'];     // get market structure by symbol
$dashcny2 = $exchange->market ('DASH/CNY');     // same result in a slightly␣
↪different way

$dashcnyId = $exchange->market_id ('DASH/CNY'); // get market id by symbol

$symbols = $exchange->symbols;                  // get an array of symbols
$symbols2 = array_keys ($exchange->markets);    // same as previous line

var_dump ($exchange->id, $symbols);             // print all symbols

$currencies = $exchange->currencies;            // a list of currencies

$okcoinusd = '\\ccxt\\okcoinusd';
$okcoinusd = new $okcoinusd ();

$okcoinusd->load_markets ();

$okcoinusd->markets['BTC/USD'];                 // symbol → market (get market by␣
↪symbol)
$okcoinusd->markets_by_id['btc_usd'];           // id → market (get market by id)

$okcoinusd->markets['BTC/USD']['id'];           // symbol → id (get id by symbol)
$okcoinusd->markets_by_id['btc_usd']['symbol']; // id → symbol (get symbol by id)
```

8.3.1 Naming Consistency

There is a bit of term ambiguity across various exchanges that may cause confusion among newcoming traders. Some exchanges call markets as *pairs*, whereas other exchanges call symbols as *products*. In terms of the ccxt library, each exchange contains one or more trading markets. Each market has an id and a symbol. Most symbols are pairs of base currency and quote currency.

`Exchanges → Markets → Symbols → Currencies`

Historically various symbolic names have been used to designate same trading pairs. Some cryptocurrencies (like Dash) even changed their names more than once during their ongoing lifetime. For consistency across exchanges the ccxt library will perform the following known substitutions for symbols and currencies:

- `XBT → BTC`: `XBT` is newer but `BTC` is more common among exchanges and sounds more like bitcoin (read more).
- `BCC → BCH`: The Bitcoin Cash fork is often called with two different symbolic names: `BCC` and `BCH`. The name `BCC` is ambiguous for Bitcoin Cash, it is confused with BitConnect. The ccxt library will convert `BCC` to `BCH` where it is appropriate (some exchanges and aggregators confuse them).
- `DRK → DASH`: `DASH` was Darkcoin then became Dash (read more).
- `DSH → DASH`: Try not to confuse symbols and currencies. The `DSH` (Dashcoin) is not the same as `DASH` (Dash). Some exchanges have `DASH` labelled inconsistently as `DSH`, the ccxt library does a correction for that as well (`DSH → DASH`), but only on certain exchanges that have these two currencies confused, whereas most exchanges have them both correct. Just remember that `DASH/BTC` is not the same as `DSH/BTC`.
- `XRB → NANO`: `NANO` is the newer code for RaiBlocks, thus, CCXT unified API uses will replace the older `XRB` with `NANO` where needed. https://hackernoon.com/nano-rebrand-announcement-9101528a7b76
- `USD → USDT`: Some exchanges, like Bitfinex, HitBTC and a few other name the currency as `USD` in their listings, but those markets are actually trading `USDT`. The confusion can come from a 3-letter limitation on symbol names or may be due to other reasons. In cases where the traded currency is actually `USDT` and is not `USD` – the CCXT library will perform `USD → USDT` conversion. Note, however, that some exchanges have both `USD` and `USDT` symbols, for example, Kraken has a `USDT/USD` trading pair.

Notes On Naming Consistency

Each exchange has an associative array of substitutions for cryptocurrency symbolic codes in the `exchange.commonCurrencies` property. Sometimes the user may notice exotic symbol names with mixed-case words and spaces in the code. The logic behind having these names is explained by the rules for resolving conflicts in naming and currency-coding when one or more currencies have the same symbolic code with different exchanges:

- First, we gather all info available from the exchanges themselves about the currency codes in question. They usually have a description of their coin listings somewhere in their API or their docs, knowledgebases or elsewhere on their websites.
- When we identify each particular cryptocurrency standing behind the currency code, we look them up on CoinMarketCap.
- The currency that has the greatest market capitalization of all wins the currency code and keeps it. For example, HOT often stand for either `Holo` or `Hydro Protocol`. In this case `Holo` retains the code `HOT`, and `Hydro Protocol` will have its name as its code, literally, `Hydro Protocol`. So, there may be trading pairs with symbols like `HOT/USD` (for `Holo`) and `Hydro Protocol/USD` – those are two different markets.
- If market cap of a particular coin is unknown or is not enough to determine the winner, we also take trading volumes and other factors into consideration.
- When the winner is determined all other competing currencies get their code names properly remapped and substituted within conflicting exchanges via `.commonCurrencies`.

- Unfortunately this is a work in progress, because new currencies get listed daily and new exchanges are added from time to time, so, in general this is a never-ending process of self-correction in a quickly changing environment, practically, in *"live mode"*. We are thankful for all reported conflicts and mismatches you may find.

Consistency Of Base And Quote Currencies

It depends on which exchange you are using, but some of them have a reversed (inconsistent) pairing of `base` and `quote`. They actually have base and quote misplaced (switched/reversed sides). In that case you'll see a difference of parsed `base` and `quote` currency values with the unparsed `info` in the market substructure.

For those exchanges the ccxt will do a correction, switching and normalizing sides of base and quote currencies when parsing exchange replies. This logic is financially and terminologically correct. If you want less confusion, remember the following rule: **base is always before the slash, quote is always after the slash in any symbol and with any market**.

```
base currency ↓
              BTC / USDT
              ETH / BTC
             DASH / ETH
                     ↑ quote currency
```

8.4 Market Cache Force Reload

The `loadMarkets () / load_markets ()` is also a dirty method with a side effect of saving the array of markets on the exchange instance. You only need to call it once per exchange. All subsequent calls to the same method will return the locally saved (cached) array of markets.

When exchange markets are loaded, you can then access market information any time via the `markets` property. This property contains an associative array of markets indexed by symbol. If you need to force reload the list of markets after you have them loaded already, pass the reload = true flag to the same method again.

```
// JavaScript
(async () => {
    let kraken = new ccxt.kraken ({ verbose: true }) // log HTTP requests
    await kraken.load_markets () // request markets
    console.log (kraken.id, kraken.markets)    // output a full list of all loaded␣
↪markets
    console.log (Object.keys (kraken.markets)) // output a short list of market␣
↪symbols
    console.log (kraken.markets['BTC/USD'])    // output single market details
    await kraken.load_markets () // return a locally cached version, no reload
    let reloadedMarkets = await kraken.load_markets (true) // force HTTP reload = true
    console.log (reloadedMarkets['ETH/BTC'])
}) ()
```

```
# Python
poloniex = ccxt.poloniex({'verbose': True}) # log HTTP requests
poloniex.load_markets() # request markets
print(poloniex.id, poloniex.markets)    # output a full list of all loaded markets
print(list(poloniex.markets.keys())) # output a short list of market symbols
print(poloniex.markets['BTC/ETH'])      # output single market details
poloniex.load_markets() # return a locally cached version, no reload
reloadedMarkets = poloniex.load_markets(True) # force HTTP reload = True
print(reloadedMarkets['ETH/ZEC'])
```

```
// PHP
$bitfinex = new \ccxt\bitfinex (array ('verbose' => true)); // log HTTP requests
$bitfinex.load_markets (); // request markets
var_dump ($bitfinex->id, $bitfinex->markets); // output a full list of all loaded␣
↪markets
var_dump (array_keys ($bitfinex->markets));   // output a short list of market symbols
var_dump ($bitfinex->markets['XRP/USD']);     // output single market details
$bitfinex->load_markets (); // return a locally cached version, no reload
$reloadedMarkets = $bitfinex->load_markets (true); // force HTTP reload = true
var_dump ($bitfinex->markets['XRP/BTC']);
```

CHAPTER 9

API Methods / Endpoints

Each exchange offers a set of API methods. Each method of the API is called an *endpoint*. Endpoints are HTTP URLs for querying various types of information. All endpoints return JSON in response to client requests.

Usually, there is an endpoint for getting a list of markets from an exchange, an endpoint for retrieving an order book for a particular market, an endpoint for retrieving trade history, endpoints for placing and canceling orders, for money deposit and withdrawal, etc... Basically every kind of action you could perform within a particular exchange has a separate endpoint URL offered by the API.

Because the set of methods differs from exchange to exchange, the ccxt library implements the following: - a public and private API for all possible URLs and methods - a unified API supporting a subset of common methods

The endpoint URLs are predefined in the `api` property for each exchange. You don't have to override it, unless you are implementing a new exchange API (at least you should know what you're doing).

9.1 Implicit API Methods

Most of exchange-specific API methods are implicit, meaning that they aren't defined explicitly anywhere in code. The library implements a declarative approach for defining implicit (non-unified) exchanges' API methods.

Each method of the API usually has its own endpoint. The library defines all endpoints for each particular exchange in the `.api` property. Upon exchange construction an implicit *magic* method (aka *partial function* or *closure*) will be created inside `defineRestApi()/define_rest_api()` on the exchange instance for each endpoint from the list of `.api` endpoints. This is performed for all exchanges universally. Each generated method will be accessible in both `camelCase` and `under_score` notations.

The endpoints definition is a **full list of ALL API URLs** exposed by an exchange. This list gets converted to callable methods upon exchange instantiation. Each URL in the API endpoint list gets a corresponding callable method. This is done automatically for all exchanges, therefore the ccxt library supports **all possible URLs** offered by crypto exchanges.

Each implicit method gets a unique name which is constructed from the `.api` definition. For example, a private HTTPS PUT `https://api.exchange.com/order/{id}/cancel` endpoint will have a corresponding exchange method named `.privatePutOrderIdCancel()/.private_put_order_id_cancel()`. A public

HTTPS GET `https://api.exchange.com/market/ticker/{pair}` endpoint would result in the corresponding method named `.publicGetTickerPair()`/`.public_get_ticker_pair()`, and so on.

An implicit method takes a dictionary of parameters, sends the request to the exchange and returns an exchange-specific JSON result from the API **as is, unparsed**. To pass a parameter, add it to the dictionary explicitly under a key equal to the parameter's name. For the examples above, this would look like `.privatePutOrderIdCancel ({ id: '41987a2b-...' })` and `.publicGetTickerPair ({ pair: 'BTC/USD' })`.

The recommended way of working with exchanges is not using exchange-specific implicit methods but using the unified ccxt methods instead. The exchange-specific methods should be used as a fallback in cases when a corresponding unified method isn't available (yet).

To get a list of all available methods with an exchange instance, including implicit methods and unified methods you can simply do the following:

```
console.log (new ccxt.kraken ())    // JavaScript
print(dir(ccxt.hitbtc()))            # Python
var_dump (new \ccxt\okcoinusd ()); // PHP
```

9.2 Public/Private API

API URLs are often grouped into two sets of methods called a *public API* for market data and a *private API* for trading and account access. These groups of API methods are usually prefixed with a word 'public' or 'private'.

A public API is used to access market data and does not require any authentication whatsoever. Most exchanges provide market data openly to all (under their rate limit). With the ccxt library anyone can access market data out of the box without having to register with the exchanges and without setting up account keys and passwords.

Public APIs include the following:

- instruments/trading pairs
- price feeds (exchange rates)
- order books (L1, L2, L3...)
- trade history (closed orders, transactions, executions)
- tickers (spot / 24h price)
- OHLCV series for charting
- other public endpoints

For trading with private API you need to obtain API keys from/to exchanges. It often means registering with exchanges and creating API keys with your account. Most exchanges require personal info or identification. Some kind of verification may be necessary as well.

If you want to trade you need to register yourself, this library will not create accounts or API keys for you. Some exchange APIs expose interface methods for registering an account from within the code itself, but most of exchanges don't. You have to sign up and create API keys with their websites.

Private APIs allow the following:

- manage personal account info
- query account balances
- trade by making market and limit orders
- create deposit addresses and fund accounts

- request withdrawal of fiat and crypto funds
- query personal open / closed orders
- query positions in margin/leverage trading
- get ledger history
- transfer funds between accounts
- use merchant services

Some exchanges offer the same logic under different names. For example, a public API is also often called *market data*, *basic*, *market*, *mapi*, *api*, *price*, etc... All of them mean a set of methods for accessing data available to public. A private API is also often called *trading*, *trade*, *tapi*, *exchange*, *account*, etc...

A few exchanges also expose a merchant API which allows you to create invoices and accept crypto and fiat payments from your clients. This kind of API is often called *merchant*, *wallet*, *payment*, *ecapi* (for e-commerce).

To get a list of all available methods with an exchange instance, you can simply do the following:

```
console.log (new ccxt.kraken ())    // JavaScript
print (dir (ccxt.hitbtc ()))         # Python
var_dump (new \ccxt\okcoinusd ()); // PHP
```

9.3 Synchronous vs Asynchronous Calls

In the JavaScript version of CCXT all methods are asynchronous and return Promises that resolve with a decoded JSON object. In CCXT we use the modern *async/await* syntax to work with Promises. If you're not familiar with that syntax, you can read more about it here.

```
// JavaScript

(async () => {
    let pairs = await kraken.publicGetSymbolsDetails ()
    let marketIds = Object.keys (pairs['result'])
    let marketId = marketIds[0]
    let ticker = await kraken.publicGetTicker ({ pair: marketId })
    console.log (kraken.id, marketId, ticker)
}) ()
```

The ccxt library supports asynchronous concurrency mode in Python 3.5+ with async/await syntax. The asynchronous Python version uses pure asyncio with aiohttp. In async mode you have all the same properties and methods, but most methods are decorated with an async keyword. If you want to use async mode, you should link against the `ccxt.async_support` subpackage, like in the following example:

```
# Python

import asyncio
import ccxt.async_support as ccxt

async def print_poloniex_ethbtc_ticker():
    poloniex = ccxt.poloniex()
    print(await poloniex.fetch_ticker('ETH/BTC'))

asyncio.get_event_loop().run_until_complete(print_poloniex_ethbtc_ticker())
```

In PHP all API methods are synchronous.

9.4 Returned JSON Objects

All public and private API methods return raw decoded JSON objects in response from the exchanges, as is, untouched. The unified API returns JSON-decoded objects in a common format and structured uniformly across all exchanges.

9.5 Passing Parameters To API Methods

The set of all possible API endpoints differs from exchange to exchange. Most of methods accept a single associative array (or a Python dict) of key-value parameters. The params are passed as follows:

```
bitso.publicGetTicker ({ book: 'eth_mxn' })                  // JavaScript
ccxt.zaif().public_get_ticker_pair ({ 'pair': 'btc_jpy' })   # Python
$luno->public_get_ticker (array ('pair' => 'XBTIDR'));       // PHP
```

For a full list of accepted method parameters for each exchange, please consult *API docs*.

9.5.1 API Method Naming Conventions

An exchange method name is a concatenated string consisting of type (public or private), HTTP method (GET, POST, PUT, DELETE) and endpoint URL path like in the following examples:

Method Name	Base API URL	Endpoint URL
publicGetIdOrderbook	https://bitbay.net/API/Public	{id}/orderbook
publicGetPairs	https://bitlish.com/api	pairs
publicGetJsonMarketTicker	https://www.bitmarket.net	json/{market}/ticker
privateGetUserMargin	https://bitmex.com	user/margin
privatePostTrade	https://btc-x.is/api	trade
tapiCancelOrder	https://yobit.net	tapi/CancelOrder
...	...	...

The ccxt library supports both camelcase notation (preferred in JavaScript) and underscore notation (preferred in Python and PHP), therefore all methods can be called in either notation or coding style in any language. Both of these notations work in JavaScript, Python and PHP:

```
exchange.methodName ()  // camelcase pseudocode
exchange.method_name () // underscore pseudocode
```

To get a list of all available methods with an exchange instance, you can simply do the following:

```
console.log (new ccxt.kraken ())    // JavaScript
print (dir (ccxt.hitbtc ()))         # Python
var_dump (new \ccxt\okcoinusd ()); // PHP
```

9.6 Unified API

The unified ccxt API is a subset of methods common among the exchanges. It currently contains the following methods:

- `fetchMarkets ()`: Fetches a list of all available markets from an exchange and returns an array of markets (objects with properties such as `symbol`, `base`, `quote` etc.). Some exchanges do not have means for obtaining a list of markets via their online API. For those, the list of markets is hardcoded.
- `loadMarkets ([reload])`: Returns the list of markets as an object indexed by symbol and caches it with the exchange instance. Returns cached markets if loaded already, unless the `reload = true` flag is forced.
- `fetchOrderBook (symbol[, limit = undefined[, params = {}]])`: Fetch L2/L3 order book for a particular market trading symbol.
- `fetchL2OrderBook (symbol[, limit = undefined[, params]])`: Level 2 (price-aggregated) order book for a particular symbol.
- `fetchTrades (symbol[, since[, [limit, [params]]]])`: Fetch recent trades for a particular trading symbol.
- `fetchTicker (symbol)`: Fetch latest ticker data by trading symbol.
- `fetchBalance ()`: Fetch Balance.
- `createOrder (symbol, type, side, amount[, price[, params]])`
- `createLimitBuyOrder (symbol, amount, price[, params])`
- `createLimitSellOrder (symbol, amount, price[, params])`
- `createMarketBuyOrder (symbol, amount[, params])`
- `createMarketSellOrder (symbol, amount[, params])`
- `cancelOrder (id[, symbol[, params]])`
- `fetchOrder (id[, symbol[, params]])`
- `fetchOrders ([symbol[, since[, limit[, params]]]])`
- `fetchOpenOrders ([symbol[, since, limit, params]]])`
- `fetchClosedOrders ([symbol[, since[, limit[, params]]]])`
- `fetchMyTrades ([symbol[, since[, limit[, params]]]])`
- ...

9.6.1 Overriding Unified API Params

Note, that most of methods of the unified API accept an optional `params` parameter. It is an associative array (a dictionary, empty by default) containing the params you want to override. The contents of `params` are exchange-specific, consult the exchanges' API documentation for supported fields and values. Use the `params` dictionary if you need to pass a custom setting or an optional parameter to your unified query.

```
// JavaScript
(async () => {

    const params = {
        'foo': 'bar',      // exchange-specific overrides in unified queries
        'Hello': 'World!', // see their docs for more details on parameter names
    }

    // the overrides go into the last argument to the unified call ↓ HERE
    const result = await exchange.fetchOrderBook (symbol, length, params)
}) ()
```

```
# Python
params = {
    'foo': 'bar',       # exchange-specific overrides in unified queries
    'Hello': 'World!',  # see their docs for more details on parameter names
}

# overrides go in the last argument to the unified call ↓ HERE
result = exchange.fetch_order_book(symbol, length, params)
```

```
// PHP
$params = array (
    'foo' => 'bar',       // exchange-specific overrides in unified queries
    'Hello' => 'World!',  // see their docs for more details on parameter names
}

// overrides go into the last argument to the unified call ↓ HERE
$result = $exchange->fetch_order_book ($symbol, $length, $params);
```

9.6.2 Pagination

Most of unified methods will return either a single object or a plain array (a list) of objects (trades, orders, transactions and so on). However, very few exchanges (if any at all) will return all orders, all trades, all ohlcv candles or all transactions at once. Most often their APIs `limit` output to a certain number of most recent objects. **YOU CANNOT GET ALL OBJECTS SINCE THE BEGINNING OF TIME TO THE PRESENT MOMENT IN JUST ONE CALL**. Practically, very few exchanges will tolerate or allow that.

To fetch historical orders or trades, the user will need to traverse the data in portions or "pages" of objects. Pagination often implies *"fetching portions of data one by one"* in a loop.

In most cases users are **required to use at least some type of pagination** in order to get the expected results consistently. If the user does not apply any pagination, most methods will return the exchanges' default, which may start from the beginning of history or may be a subset of most recent objects. The default behaviour (without pagination) is exchange-specific! The means of pagination are often used with the following methods in particular:

- `fetchTrades`
- `fetchOHLCV`
- `fetchOrders`
- `fetchOpenOrders`
- `fetchClosedOrders`
- `fetchMyTrades`
- `fetchTransactions`
- `fetchDeposits`
- `fetchWithdrawals`

With methods returning lists of objects, exchanges may offer one or more types of pagination. CCXT unifies **date-based pagination** by default, with timestamps **in milliseconds** throughout the entire library.

Working With Datetimes and Timestamps

The set of methods for working with UTC dates and timestamps and for converting between them:

```
exchange.parse8601 ('2018-01-01T00:00:00Z') == 1514764800000 // integer, Z = UTC
exchange.iso8601 (1514764800000) == '2018-01-01T00:00:00Z'   // iso8601 string
exchange.seconds ()      // integer UTC timestamp in seconds
exchange.milliseconds () // integer UTC timestamp in milliseconds
```

Date-based pagination

This is the type of pagination currently used throughout the CCXT Unified API. The user supplies a `since` timestamp **in milliseconds** (!) and a number to `limit` results. To traverse the objects of interest page by page, the user runs the following (below is pseudocode, it may require overriding some exchange-specific params, depending on the exchange in question):

```
// JavaScript
if (exchange.has['fetchTrades']) {
    let since = exchange.milliseconds () - 86400000 // -1 day from now
    // alternatively, fetch from a certain starting datetime
    // let since = exchange.parse8601 ('2018-01-01T00:00:00Z')
    let allTrades = []
    while (since < exchange.milliseconds ()) {
        const symbol = undefined // change for your symbol
        const limit = 20 // change for your limit
        const trades = await exchange.fetchTrades (symbol, since, limit)
        if (trades.length) {
            since = trades[trades.length - 1]
            allTrades.push (trades)
        } else {
            break
        }
    }
}
```

```
# Python
if exchange.has['fetchOrders']:
    since = exchange.milliseconds () - 86400000  # -1 day from now
    # alternatively, fetch from a certain starting datetime
    # since = exchange.parse8601('2018-01-01T00:00:00Z')
    all_orders = []
    while since < exchange.milliseconds ():
        symbol = None  # change for your symbol
        limit = 20  # change for your limit
        orders = await exchange.fetch_orders(symbol, since, limit)
        if len(orders):
            since = orders[len(orders) - 1]
            all_orders += orders
        else:
            break
```

```
// PHP
if ($exchange->has['fetchMyTrades']) {
    $since = exchange->milliseconds () - 86400000; // -1 day from now
    // alternatively, fetch from a certain starting datetime
    // $since = $exchange->parse8601 ('2018-01-01T00:00:00Z')
    $all_trades = array ();
    while (since < exchange->milliseconds ()) {
        $symbol = null; // change for your symbol
```

(continues on next page)

(continued from previous page)

```
        $limit = 20; // change for your limit
        $trades = $exchange->fetchMyTrades ($symbol, $since, $limit);
        if (count($trades)) {
            $since = $trades[count($trades) - 1];
            $all_trades = array_merge ($all_trades, $trades);
        } else {
            break;
        }
    }
}
```

id-based pagination

The user supplies a `from_id` of the object, from where the query should continue returning results, and a number to `limit` results. This is the default with some exchanges, however, this type is not unified (yet). To paginate objects based on their ids, the user would run the following:

```
// JavaScript
if (exchange.has['fetchTrades']) {
    let from_id = 'abc123' // all ids are strings
    let allTrades = []
    while (true) {
        const symbol = undefined // change for your symbol
        const since = undefined
        const limit = 20 // change for your limit
        const params = {
            'from_id': from_id, // exchange-specific non-unified parameter name
        }
        const trades = await exchange.fetchTrades (symbol, since, limit, params)
        if (trades.length) {
            from_id = trades[trades.length - 1]['id']
            allTrades.push (trades)
        } else {
            break
        }
    }
}
```

```
# Python
if exchange.has['fetchOrders']:
    from_id = 'abc123'  # all ids are strings
    all_orders = []
    while True:
        symbol = None  # change for your symbol
        since = None
        limit = 20  # change for your limit
        params = {
            'from_id': from_id,  # exchange-specific non-unified parameter name
        }
        orders = await exchange.fetch_orders(symbol, since, limit, params)
        if len(orders):
            from_id = orders[len(orders) - 1]['id']
            all_orders += orders
        else:
            break
```

```
// PHP
if ($exchange->has['fetchMyTrades']) {
    $from_id = 'abc123' // all ids are strings
    $all_trades = array ();
    while (true) {
        $symbol = null; // change for your symbol
        $since = null;
        $limit = 20; // change for your limit
        $params = array (
            'from_id' => $from_id, // exchange-specific non-unified parameter name
        );
        $trades = $exchange->fetchMyTrades ($symbol, $since, $limit, $params);
        if (count($trades)) {
            $from_id = $trades[count($trades) - 1]['id'];
            $all_trades = array_merge ($all_trades, $trades);
        } else {
            break;
        }
    }
}
```

Pagenumber-based (cursor) pagination

The user supplies a page number or an *initial "cursor"* value. The exchange returns a page of results and the *next "cursor"* value, to proceed from. Most of exchanges that implement this type of pagination will either return the next cursor within the response itself or will return the next cursor values within HTTP response headers.

See an example implementation here: https://github.com/ccxt/ccxt/blob/master/examples/py/gdax-fetch-my-trades-pagination.py

Upon each iteration of the loop the user has to take the next cursor and put it into the overrided params for the next query (on the following iteration):

```
// JavaScript
if (exchange.has['fetchTrades']) {
    let page = 0  // exchange-specific type and value
    let allTrades = []
    while (true) {
        const symbol = undefined // change for your symbol
        const since = undefined
        const limit = 20 // change for your limit
        const params = {
            'page': page, // exchange-specific non-unified parameter name
        }
        const trades = await exchange.fetchTrades (symbol, since, limit, params)
        if (trades.length) {
            // not thread-safu and exchange-specific !
            page = exchange.last_json_response['cursor']
            allTrades.push (trades)
        } else {
            break
        }
    }
}
```

```python
# Python
if exchange.has['fetchOrders']:
    cursor = 0  # exchange-specific type and value
    all_orders = []
    while True:
        symbol = None  # change for your symbol
        since = None
        limit = 20  # change for your limit
        params = {
            'cursor': cursor,  # exchange-specific non-unified parameter name
        }
        orders = await exchange.fetch_orders(symbol, since, limit, params)
        if len(orders):
            # not thread-safu and exchange-specific !
            cursor = exchange.last_http_headers['CB-AFTER']
            all_orders += orders
        else:
            break
```

```php
// PHP
if ($exchange->has['fetchMyTrades']) {
    $start = '0' // exchange-specific type and value
    $all_trades = array ();
    while (true) {
        $symbol = null; // change for your symbol
        $since = null;
        $limit = 20; // change for your limit
        $params = array (
            'start' => $start, // exchange-specific non-unified parameter name
        );
        $trades = $exchange->fetchMyTrades ($symbol, $since, $limit, $params);
        if (count($trades)) {
            // not thread-safu and exchange-specific !
            $start = $exchange->last_json_response['next'];
            $all_trades = array_merge ($all_trades, $trades);
        } else {
            break;
        }
    }
}
```

CHAPTER 10

Market Data

- Order Book / Market Depth
  - Market Price
- Price Tickers
  - Individually By Symbol
  - All At Once
- OHLCV Candlestick Charts
- Public Trades

10.1 Order Book

Exchanges expose information on open orders with bid (buy) and ask (sell) prices, volumes and other data. Usually there is a separate endpoint for querying current state (stack frame) of the *order book* for a particular market. An order book is also often called *market depth*. The order book information is used in the trading decision making process.

The method for fetching an order book for a particular symbol is named `fetchOrderBook` or `fetch_order_book`. It accepts a symbol and an optional dictionary with extra params (if supported by a particular exchange). The method for fetching the order book is called like shown below:

```
// JavaScript
delay = 2000 // milliseconds = seconds * 1000
(async () => {
    for (symbol in exchange.markets) {
        console.log (await exchange.fetchOrderBook (symbol))
        await new Promise (resolve => setTimeout (resolve, delay)) // rate limit
    }
}) ()
```

```
# Python
import time
delay = 2 # seconds
for symbol in exchange.markets:
    print (exchange.fetch_order_book (symbol))
    time.sleep (delay) # rate limit
```

```
// PHP
$delay = 2000000; // microseconds = seconds * 1000000
foreach ($exchange->markets as $symbol => $market) {
    var_dump ($exchange->fetch_order_book ($symbol));
    usleep ($delay); // rate limit
}
```

The structure of a returned order book is as follows:

```
{
    'bids': [
        [ price, amount ], // [ float, float ]
        [ price, amount ],
        ...
    ],
    'asks': [
        [ price, amount ],
        [ price, amount ],
        ...
    ],
    'timestamp': 1499280391811, // Unix Timestamp in milliseconds (seconds * 1000)
    'datetime': '2017-07-05T18:47:14.692Z', // ISO8601 datetime string with␣
↪milliseconds
}
```

The timestamp and datetime may be missing (‘‘undefined/None/null‘‘) if the exchange in question does not provide a corresponding value in the API response.

Prices and amounts are floats. The bids array is sorted by price in descending order. The best (highest) bid price is the first element and the worst (lowest) bid price is the last element. The asks array is sorted by price in ascending order. The best (lowest) ask price is the first element and the worst (highest) ask price is the last element. Bid/ask arrays can be empty if there are no corresponding orders in the order book of an exchange.

Exchanges may return the stack of orders in various levels of details for analysis. It is either in full detail containing each and every order, or it is aggregated having slightly less detail where orders are grouped and merged by price and volume. Having greater detail requires more traffic and bandwidth and is slower in general but gives a benefit of higher precision. Having less detail is usually faster, but may not be enough in some very specific cases.

10.1.1 Notes On Order Book Structure

- `orderbook['timestamp']` is the time when the exchange generated this orderbook response (before replying it back to you). This may be missing (`undefined/None/null`), as documented in the Manual, not all exchanges provide a timestamp there. If it is defined, then it is the UTC timestamp **in milliseconds** since 1 Jan 1970 00:00:00.
- `exchange.last_response_headers['Date']` is the date-time string of the last HTTP response received (from HTTP headers). The ‘Date’ parser should respect the timezone designated there. The precision of the date-time is 1 second, 1000 milliseconds. This date should be set by the exchange server when the message originated according to the following standards:

- https://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.18
- https://tools.ietf.org/html/rfc1123#section-5.2.14
- https://tools.ietf.org/html/rfc822#section-5

10.1.2 Market Depth

Some exchanges accept a dictionary of extra parameters to the `fetchOrderBook () / fetch_order_book ()` function. **All extra ''params'' are exchange-specific (non-unified)**. You will need to consult exchanges docs if you want to override a particular param, like the depth of the order book. You can get a limited count of returned orders or a desired level of aggregation (aka *market depth*) by specifying an limit argument and exchange-specific extra `params` like so:

```
// JavaScript

(async function test () {
    const ccxt = require ('ccxt')
    const exchange = new ccxt.bitfinex ()
    const limit = 5
    const orders = await exchange.fetchOrderBook ('BTC/USD', limit, {
        // this parameter is exchange-specific, all extra params have unique names␣
↪per exchange
        'group': 1, // 1 = orders are grouped by price, 0 = orders are separate
    })
}) ()
```

```
# Python

import ccxt
# return up to ten bidasks on each side of the order book stack
limit = 10
ccxt.cex().fetch_order_book('BTC/USD', limit)
```

```
// PHP

// instantiate the exchange by id
$exchange = '\\ccxt\\kraken';
$exchange = new $exchange ();
// up to ten orders on each side, for example
$limit = 20;
var_dump ($exchange->fetch_order_book ('BTC/USD', $limit));
```

The levels of detail or levels of order book aggregation are often number-labelled like L1, L2, L3...

- **L1**: less detail for quickly obtaining very basic info, namely, the market price only. It appears to look like just one order in the order book.
- **L2**: most common level of aggregation where order volumes are grouped by price. If two orders have the same price, they appear as one single order for a volume equal to their total sum. This is most likely the level of aggregation you need for the majority of purposes.
- **L3**: most detailed level with no aggregation where each order is separate from other orders. This LOD naturally contains duplicates in the output. So, if two orders have equal prices they are **not** merged together and it's up to the exchange's matching engine to decide on their priority in the stack. You don't really need L3 detail for successful trading. In fact, you most probably don't need it at all. Therefore some exchanges don't support it and always return aggregated order books.

If you want to get an L2 order book, whatever the exchange returns, use the `fetchL2OrderBook(symbol, limit, params)` or `fetch_l2_order_book(symbol, limit, params)` unified method for that.

10.1.3 Market Price

In order to get current best price (query market price) and calculate bidask spread take first elements from bid and ask, like so:

```
// JavaScript
let orderbook = exchange.fetchOrderBook (exchange.symbols[0])
let bid = orderbook.bids.length ? orderbook.bids[0][0] : undefined
let ask = orderbook.asks.length ? orderbook.asks[0][0] : undefined
let spread = (bid && ask) ? ask - bid : undefined
console.log (exchange.id, 'market price', { bid, ask, spread })
```

```
# Python
orderbook = exchange.fetch_order_book (exchange.symbols[0])
bid = orderbook['bids'][0][0] if len (orderbook['bids']) > 0 else None
ask = orderbook['asks'][0][0] if len (orderbook['asks']) > 0 else None
spread = (ask - bid) if (bid and ask) else None
print (exchange.id, 'market price', { 'bid': bid, 'ask': ask, 'spread': spread })
```

```
// PHP
$orderbook = $exchange->fetch_order_book ($exchange->symbols[0]);
$bid = count ($orderbook['bids']) ? $orderbook['bids'][0][0] : null;
$ask = count ($orderbook['asks']) ? $orderbook['asks'][0][0] : null;
$spread = ($bid && $ask) ? $ask - $bid : null;
$result = array ('bid' => $bid, 'ask' => $ask, 'spread' => $spread);
var_dump ($exchange->id, 'market price', $result);
```

10.2 Price Tickers

A price ticker contains statistics for a particular market/symbol for some period of time in recent past, usually last 24 hours. The structure of a ticker is as follows:

```
{
    'symbol':        string symbol of the market ('BTC/USD', 'ETH/BTC', ...)
    'info':        { the original non-modified unparsed reply from exchange API },
    'timestamp':     int (64-bit Unix Timestamp in milliseconds since Epoch 1 Jan␣
↪1970)
    'datetime':      ISO8601 datetime string with milliseconds
    'high':          float, // highest price
    'low':           float, // lowest price
    'bid':           float, // current best bid (buy) price
    'bidVolume':     float, // current best bid (buy) amount (may be missing or␣
↪undefined)
    'ask':           float, // current best ask (sell) price
    'askVolume':     float, // current best ask (sell) amount (may be missing or␣
↪undefined)
    'vwap':          float, // volume weighed average price
    'open':          float, // opening price
    'close':         float, // price of last trade (closing price for current period)
    'last':          float, // same as `close`, duplicated for convenience
```

(continues on next page)

(continued from previous page)

```
    'previousClose': float, // closing price for the previous period
    'change':        float, // absolute change, `last - open`
    'percentage':    float, // relative change, `(change/open) * 100`
    'average':       float, // average price, `(last + open) / 2`
    'baseVolume':    float, // volume of base currency traded for last 24 hours
    'quoteVolume':   float, // volume of quote currency traded for last 24 hours
}
```

- The `bidVolume` is the volume (amount) of current best bid in the orderbook.
- The `askVolume` is the volume (amount) of current best ask in the orderbook.
- The `baseVolume` is the amount of base currency traded (bought or sold) in last 24 hours.
- The `quoteVolume` is the amount of quote currency traded (bought or sold) in last 24 hours.

All prices in ticker structure are in quote currency. Some fields in a returned ticker structure may be undefined/None/null.

```
base currency ↓
             BTC / USDT
             ETH / BTC
            DASH / ETH
                    ↑ quote currency
```

Timestamp and datetime are both Universal Time Coordinated (UTC) in milliseconds.

Although some exchanges do mix-in orderbook's top bid/ask prices into their tickers (and some even top bid/asks volumes) you should not treat ticker as a `fetchOrderBook` replacement. The main purpose of a ticker is to serve statistical data, as such, treat it as "live 24h OHLCV". It is known that exchanges discourage frequent `fetchTicker` requests by imposing stricter rate limits on these queries. If you need a unified way to access bid/asks you should use `fetchL[123]OrderBook` family instead.

To get historical prices and volumes use the unified `fetchOHLCV <https://github.com/ccxt/ccxt/wiki/Manual#ohlcv-candlestick-charts>`__ method where available.

10.2.1 Individually By Symbol

To get the individual ticker data from an exchange for each particular trading pair or symbol call the `fetchTicker (symbol)`:

```
// JavaScript
if (exchange.has['fetchTicker']) {
    console.log (await (exchange.fetchTicker ('BTC/USD'))) // ticker for BTC/USD
    let symbols = Object.keys (exchange.markets)
    let random = Math.floor (Math.random () * (symbols.length - 1))
    console.log (exchange.fetchTicker (symbols[random])) // ticker for a random symbol
}
```

```
# Python
import random
if (exchange.has['fetchTicker']):
    print(exchange.fetch_ticker('LTC/ZEC')) # ticker for LTC/ZEC
    symbols = list(exchange.markets.keys())
    print(exchange.fetch_ticker(random.choice(symbols))) # ticker for a random symbol
```

```
// PHP (don't forget to set your timezone properly!)
if ($exchange->has['fetchTicker']) {
    var_dump ($exchange->fetch_ticker ('ETH/CNY')); // ticker for ETH/CNY
    $symbols = array_keys ($exchange->markets);
    $random = rand () % count ($symbols);
    var_dump ($exchange->fetch_ticker ($symbols[$random])); // ticker for a random␣
↪symbol
}
```

10.2.2 All At Once

Some exchanges (not all of them) also support fetching all tickers at once. See their docs for details. You can fetch all tickers with a single call like so:

```
// JavaScript
if (exchange.has['fetchTickers']) {
    console.log (await (exchange.fetchTickers ())) // all tickers indexed by their␣
↪symbols
}
```

```
# Python
if (exchange.has['fetchTickers']):
    print(exchange.fetch_tickers()) # all tickers indexed by their symbols
```

```
// PHP
if ($exchange->has['fetchTickers']) {
    var_dump ($exchange->fetch_tickers ()); // all tickers indexed by their symbols
}
```

Fetching all tickers requires more traffic than fetching a single ticker. If you only need one ticker, fetching by a particular symbol is faster in general. You probably want to fetch all tickers only if you really need all of them.

The structure of returned value is as follows:

```
{
    'info':    { ... }, // the original JSON response from the exchange as is
    'BTC/USD': { ... }, // a single ticker for BTC/USD
    'ETH/BTC': { ... }, // a ticker for ETH/BTC
    ...
}
```

A general solution for fetching all tickers from all exchanges (even the ones that don't have a corresponding API endpoint) is on the way, this section will be updated soon.

```
UNDER CONSTRUCTION
```

Async Mode / Concurrency

```
UNDER CONSTRUCTION
```

10.3 OHLCV Candlestick Charts

```
- this is under heavy development right now, contributions appreciated
```

Most exchanges have endpoints for fetching OHLCV data, but some of them don't. The exchange boolean (true/false) property named `has['fetchOHLCV']` indicates whether the exchange supports candlestick data series or not.

The `fetchOHLCV` method is declared in the following way:

```
fetchOHLCV (symbol, timeframe = '1m', since = undefined, limit = undefined, params =
↪{})
```

You can call the unified `fetchOHLCV` / `fetch_ohlcv` method to get the list of OHLCV candles for a particular symbol like so:

```
// JavaScript
let sleep = (ms) => new Promise (resolve => setTimeout (resolve, ms));
if (exchange.has.fetchOHLCV) {
    for (symbol in exchange.markets) {
        await sleep (exchange.rateLimit) // milliseconds
        console.log (await exchange.fetchOHLCV (symbol, '1m')) // one minute
    }
}
```

```
# Python
import time
if exchange.has['fetchOHLCV']:
    for symbol in exchange.markets:
        time.sleep (exchange.rateLimit / 1000) # time.sleep wants seconds
        print (symbol, exchange.fetch_ohlcv (symbol, '1d')) # one day
```

```
// PHP
if ($exchange->has['fetchOHLCV']) {
    foreach ($exchange->markets as $symbol => $market) {
        usleep ($exchange->rateLimit * 1000); // usleep wants microseconds
        var_dump ($exchange->fetch_ohlcv ($symbol, '1M')); // one month
    }
}
```

To get the list of available timeframes for your exchange see the `timeframes` property. Note that it is only populated when `has['fetchOHLCV']` is true as well.

There's a limit on how far back in time your requests can go. Most of exchanges will not allow to query detailed candlestick history (like those for 1-minute and 5-minute timeframes) too far in the past. They usually keep a reasonable amount of most recent candles, like 1000 last candles for any timeframe is more than enough for most of needs. You can work around that limitation by continuously fetching (aka *REST polling*) latest OHLCVs and storing them in a CSV file or in a database.

Note that the info from the last (current) candle may be incomplete until the candle is closed (until the next candle starts).

Like with most other unified and implicit methods, the `fetchOHLCV` method accepts as its last argument an associative array (a dictionary) of extra `params`, which is used to override default values that are sent in requests to the exchanges. The contents of `params` are exchange-specific, consult the exchanges' API documentation for supported fields and values.

The `since` argument is an integer UTC timestamp **in milliseconds** (everywhere throughout the library with all unified methods).

If `since` is not specified the `fetchOHLCV` method will return the time range as is the default from the exchange itself. This is not a bug. Some exchanges will return candles from the beginning of time, others will return most recent candles only, the exchanges' default behaviour is expected. Thus, without specifying `since` the range of returned candles will be exchange-specific. One should pass the `since` argument to ensure getting precisely the history range needed.

10.3.1 OHLCV Structure

The fetchOHLCV method shown above returns a list (a flat array) of OHLCV candles represented by the following structure:

```
[
    [
        1504541580000, // UTC timestamp in milliseconds, integer
        4235.4,        // (O)pen price, float
        4240.6,        // (H)ighest price, float
        4230.0,        // (L)owest price, float
        4230.7,        // (C)losing price, float
        37.72941911    // (V)olume (in terms of the base currency), float
    ],
    ...
]
```

The list of candles is returned sorted in ascending (historical) order, oldest candle first, most recent candle last.

10.3.2 OHLCV Emulation

Some exchanges don't offer any OHLCV method, and for those, the ccxt library will emulate OHLCV candles from Public Trades. In that case you will see `exchange.has['fetchOHLCV'] = 'emulated'`. However, because the trade history is usually very limited, the emulated fetchOHLCV methods cover most recent info only and should only be used as a fallback, when no other option is available.

WARNING: the fetchOHLCV emulation is experimental!

10.4 Trades, Executions, Transactions

```
- this is under heavy development right now, contributions appreciated
```

You can call the unified `fetchTrades` / `fetch_trades` method to get the list of most recent trades for a particular symbol. The `fetchTrades` method is declared in the following way:

```
async fetchTrades (symbol, since = undefined, limit = undefined, params = {})
```

For example, if you want to print recent trades for all symbols one by one sequentially (mind the rateLimit!) you would do it like so:

```
// JavaScript
if (exchange.has['fetchTrades']) {
    let sleep = (ms) => new Promise (resolve => setTimeout (resolve, ms));
    for (symbol in exchange.markets) {
        await sleep (exchange.rateLimit) // milliseconds
        console.log (await exchange.fetchTrades (symbol))
```

(continues on next page)

(continued from previous page)

```
    }
}
```

```
# Python
import time
if exchange.has['fetchTrades']:
    for symbol in exchange.markets:  # ensure you have called loadMarkets() or load_
↪markets() method.
        time.sleep (exchange.rateLimit / 1000)  # time.sleep wants seconds
        print (symbol, exchange.fetch_trades (symbol))
```

```
// PHP
if ($exchange->has['fetchTrades']) {
    foreach ($exchange->markets as $symbol => $market) {
        usleep ($exchange->rateLimit * 1000); // usleep wants microseconds
        var_dump ($exchange->fetch_trades ($symbol));
    }
}
```

The fetchTrades method shown above returns an ordered list of trades (a flat array, sorted by timestamp in ascending order, oldest trade first, most recent trade last). A list of trades is represented by the following structure:

```
[
    {
        'info':       { ... },                  // the original decoded JSON as is
        'id':        '12345-67890:09876/54321', // string trade id
        'timestamp':  1502962946216,            // Unix timestamp in milliseconds
        'datetime':  '2017-08-17 12:42:48.000', // ISO8601 datetime with milliseconds
        'symbol':    'ETH/BTC',                 // symbol
        'order':     '12345-67890:09876/54321', // string order id or undefined/None/
↪null
        'type':      'limit',                   // order type, 'market', 'limit' or␣
↪undefined/None/null
        'side':      'buy',                     // direction of the trade, 'buy' or
↪'sell'
        'price':      0.06917684,               // float price in quote currency
        'amount':     1.5,                      // amount of base currency
    },
    ...
]
```

Most exchanges return most of the above fields for each trade, though there are exchanges that don't return the type, the side, the trade id or the order id of the trade. Most of the time you are guaranteed to have the timestamp, the datetime, the symbol, the price and the amount of each trade.

The second optional argument `since` reduces the array by timestamp, the third `limit` argument reduces by number (count) of returned items.

If the user does not specify `since`, the `fetchTrades` method will return the default range of public trades from the exchange. The default set is exchange-specific, some exchanges will return trades starting from the date of listing a pair on the exchange, other exchanges will return a reduced set of trades (like, last 24 hours, last 100 trades, etc). If the user wants precise control over the timeframe, the user is responsible for specifying the `since` argument.

The `fetchTrades ()` / `fetch_trades()` method also accepts an optional `params` (assoc-key array/dict, empty by default) as its fourth argument. You can use it to pass extra params to method calls or to override a particular default value (where supported by the exchange). See the API docs for your exchange for more details.

```
UNDER CONSTRUCTION
```

CHAPTER 11

Trading

In order to be able to access your user account, perform algorithmic trading by placing market and limit orders, query balances, deposit and withdraw funds and so on, you need to obtain your API keys for authentication from each exchange you want to trade with. They usually have it available on a separate tab or page within your user account settings. API keys are exchange-specific and cannnot be interchanged under any circumstances.

11.1 Authentication

Authentication with all exchanges is handled automatically if provided with proper API keys. The process of authentication usually goes through the following pattern:

1. Generate new nonce. A nonce is an integer, often a Unix Timestamp in seconds or milliseconds (since epoch January 1, 1970). The nonce should be unique to a particular request and constantly increasing, so that no two requests share the same nonce. Each next request should have greater nonce than the previous request. **The default nonce is a 32-bit Unix Timestamp in seconds.**
2. Append public apiKey and nonce to other endpoint params, if any, then serialize the whole thing for signing.
3. Sign the serialized params using HMAC-SHA256/384/512 or MD5 with your secret key.
4. Append the signature in Hex or Base64 and nonce to HTTP headers or body.

This process may differ from exchange to exchange. Some exchanges may want the signature in a different encoding, some of them vary in header and body param names and formats, but the general pattern is the same for all of them.

You should not share the same API keypair across multiple instances of an exchange running simultaneously, in separate scripts or in multiple threads. Using the same keypair from different instances simultaneously may cause all sorts of unexpected behaviour.

The authentication is already handled for you, so you don't need to perform any of those steps manually unless you are implementing a new exchange class. The only thing you need for trading is the actual API key pair.

11.2 API Keys Setup

The API credentials usually include the following:

- `apiKey`. This is your public API Key and/or Token. This part is *non-secret*, it is included in your request header or body and sent over HTTPS in open text to identify your request. It is often a string in Hex or Base64 encoding or an UUID identifier.
- `secret`. This is your private key. Keep it secret, don't tell it to anybody. It is used to sign your requests locally before sending them to exchanges. The secret key does not get sent over the internet in the request-response process and should not be published or emailed. It is used together with the nonce to generate a cryptographically strong signature. That signature is sent with your public key to authenticate your identity. Each request has a unique nonce and therefore a unique cryptographic signature.
- `uid`. Some exchanges (not all of them) also generate a user id or *uid* for short. It can be a string or numeric literal. You should set it, if that is explicitly required by your exchange. See their docs for details.
- `password`. Some exchanges (not all of them) also require your password/phrase for trading. You should set this string, if that is explicitly required by your exchange. See their docs for details.

In order to create API keys find the API tab or button in your user settings on the exchange website. Then create your keys and copy-paste them to your config file. Your config file permissions should be set appropriately, unreadable to anyone except the owner.

Remember to keep your apiKey and secret key safe from unauthorized use, do not send or tell it to anybody. A leak of the secret key or a breach in security can cost you a fund loss.

To set up an exchange for trading just assign the API credentials to an existing exchange instance or pass them to exchange constructor upon instantiation, like so:

```
// JavaScript

const ccxt = require ('ccxt')

// any time
let kraken = new ccxt.kraken ()
kraken.apiKey = 'YOUR_KRAKEN_API_KEY'
kraken.secret = 'YOUR_KRAKEN_SECRET_KEY'

// upon instantiation
let okcoinusd = new ccxt.okcoinusd ({
    apiKey: 'YOUR_OKCOIN_API_KEY',
    secret: 'YOUR_OKCOIN_SECRET_KEY',
})

// from variable id
const exchangeId = 'binance'
    , exchangeClass = ccxt[exchangeId]
    , exchange = new exchangeClass ({
        'apiKey': 'YOUR_API_KEY',
        'secret': 'YOUR_SECRET',
        'timeout': 30000,
        'enableRateLimit': true,
    })
```

```
# Python

import ccxt
```

(continues on next page)

(continued from previous page)

```
# any time
bitfinex = ccxt.bitfinex ()
bitfinex.apiKey = 'YOUR_BFX_API_KEY'
bitfinex.secret = 'YOUR_BFX_SECRET'

# upon instantiation
hitbtc = ccxt.hitbtc ({
    'apiKey': 'YOUR_HITBTC_API_KEY',
    'secret': 'YOUR_HITBTC_SECRET_KEY',
})

# from variable id
exchange_id = 'binance'
exchange_class = getattr(ccxt, exchange_id)
exchange = exchange_class({
    'apiKey': 'YOUR_API_KEY',
    'secret': 'YOUR_SECRET',
    'timeout': 30000,
    'enableRateLimit': True,
})
```

```
// PHP

include 'ccxt.php'

// any time
$quoinex = new \ccxt\quoinex ();
$quoinex->apiKey = 'YOUR_QUOINE_API_KEY';
$quoinex->secret = 'YOUR_QUOINE_SECRET_KEY';

// upon instantiation
$zaif = new \ccxt\zaif (array (
    'apiKey' => 'YOUR_ZAIF_API_KEY',
    'secret' => 'YOUR_ZAIF_SECRET_KEY'
));

// from variable id
$exchange_id = 'binance';
$exchange_class = "\\ccxt\\$exchange_id";
$exchange = new $exchange_class (array (
    'apiKey' => 'YOUR_API_KEY',
    'secret' => 'YOUR_SECRET',
    'timeout' => 30000,
    'enableRateLimit' => true,
));
```

Note that your private requests will fail with an exception or error if you don't set up your API credentials before you start trading. To avoid character escaping **always write your credentials in single quotes**, not double quotes (`'VERY_GOOD'`, `"VERY_BAD"`).

11.3 Querying Account Balance

The returned balance structure is as follows:

```
{
    'info':  { ... },    // the original untouched non-parsed reply with details

    //------------------------------------------------------------------------------
    // indexed by availability of funds first, then by currency

    'free':  {           // money, available for trading, by currency
        'BTC': 321.00,   // floats...
        'USD': 123.00,
        ...
    },

    'used':  { ... },    // money on hold, locked, frozen, or pending, by currency

    'total': { ... },    // total (free + used), by currency

    //------------------------------------------------------------------------------
    // indexed by currency first, then by availability of funds

    'BTC':   {           // string, three-letter currency code, uppercase
        'free': 321.00   // float, money available for trading
        'used': 234.00,  // float, money on hold, locked, frozen or pending
        'total': 555.00, // float, total balance (free + used)
    },

    'USD':   {           // ...
        'free': 123.00   // ...
        'used': 456.00,
        'total': 579.00,
    },

    ...
}
```

Some exchanges may not return full balance info. Many exchanges do not return balances for your empty or unused accounts. In that case some currencies may be missing in returned balance structure.

```
// JavaScript
(async () => {
    console.log (await exchange.fetchBalance ())
}) ()
```

```
# Python
print (exchange.fetch_balance ())
```

```
// PHP
var_dump ($exchange->fetch_balance ());
```

11.3.1 Balance inference

Some exchanges do not return the full set of balance information from their API. Those will only return just the `free` or just the `total` funds, i.e. funds `used` on orders unknown. In such cases ccxt will try to obtain the missing data from *.orders cache* and will guess complete balance info from what is known for sure. However, in rare cases the available info may not be enough to deduce the missing part, thus, **the user shoud be aware of the possibility of not getting complete balance info from less sophisticated exchanges**.

11.4 Orders

```
- this part of the unified API is currenty a work in progress
- there may be some issues and missing implementations here and there
- contributions, pull requests and feedback appreciated
```

11.4.1 Querying Orders

Most of the time you can query orders by an id or by a symbol, though not all exchanges offer a full and flexible set of endpoints for querying orders. Some exchanges might not have a method for fetching recently closed orders, the other can lack a method for getting an order by id, etc. The ccxt library will target those cases by making workarounds where possible.

The list of methods for querying orders consists of the following:

- `fetchOrder (id, symbol = undefined, params = {})`
- `fetchOrders (symbol = undefined, since = undefined, limit = undefined, params = {})`
- `fetchOpenOrders (symbol = undefined, since = undefined, limit = undefined, params = {})`
- `fetchClosedOrders (symbol = undefined, since = undefined, limit = undefined, params = {})`

Note that the naming of those methods indicates if the method returns a single order or multiple orders (an array/list of orders). The `fetchOrder()` method requires a mandatory order id argument (a string). Some exchanges also require a symbol to fetch an order by id, where order ids can intersect with various trading pairs. Also, note that all other methods above return an array (a list) of orders. Most of them will require a symbol argument as well, however, some exchanges allow querying with a symbol unspecified (meaning *all symbols*).

The library will throw a NotSupported exception if a user calls a method that is not available from the exchange or is not implemented in ccxt.

To check if any of the above methods are available, look into the `.has` property of the exchange:

```
// JavaScript
'use strict';

const ccxt = require ('ccxt')
const id = 'poloniex'
exchange = new ccxt[id] ()
console.log (exchange.has)
```

```
# Python
import ccxt
id = 'cryptopia'
exchange = getattr(ccxt, 'id') ()
print(exchange.has)
```

```
// PHP
$exchange = new \ccxt\liqui ();
print_r ($exchange->has); // or var_dump
```

A typical structure of the `.has` property usually contains the following flags corresponding to order API methods for querying orders:

```
exchange.has = {

    // ... other flags ...

    'fetchOrder': true, // available from the exchange directly and implemented in␣
↪ccxt
    'fetchOrders': false, // not available from the exchange or not implemented in␣
↪ccxt
    'fetchOpenOrders': true,
    'fetchClosedOrders': 'emulated', // not available from the exchange, but emulated␣
↪in ccxt

    // ... other flags ...

}
```

The meanings of boolean `true` and `false` are obvious. A string value of `emulated` means that particular method is missing in the exchange API and ccxt will workaround that where possible by adding a caching layer, the `.orders` cache. The next section describes the inner workings of the `.orders` cache, one has to understand it to do order management with ccxt effectively.

Querying Multiple Orders And Trades

All methods returning lists of trades and lists of orders, accept the second `since` argument and the third `limit` argument:

- `fetchTrades` (public)
- `fetchMyTrades` (private)
- `fetchOrders`
- `fetchOpenOrders`
- `fetchClosedOrders`

The second argument `since` reduces the array by timestamp, the third `limit` argument reduces by number (count) of returned items.

If the user does not specify `since`, the `fetchTrades/fetchOrders` method will return the default set from the exchange. The default set is exchange-specific, some exchanges will return trades or recent orders starting from the date of listing a pair on the exchange, other exchanges will return a reduced set of trades or orders (like, last 24 hours, last 100 trades, first 100 orders, etc). If the user wants precise control over the timeframe, the user is responsible for specifying the `since` argument.

NOTE: not all exchanges provide means for filtering the lists of trades and orders by starting time, so, the support for ‘‘since‘‘ and ‘‘limit‘‘ is exchange-specific. However, most exchanges do provide at least some alternative for “pagination” and “scrolling” which can be overrided with extra ‘‘params‘‘ argument.

`.orders` cache

Some exchanges do not have a method for fetching closed orders or all orders. They will offer just the `fetchOpenOrders` endpoint, sometimes they are also generous to offer a `fetchOrder` endpoint as well. This means that they don’t have any methods for fetching the order history. The ccxt library will try to emulate the order history for the user by keeping the cached .orders property containing all orders issued within a particular exchange class instance.

Whenever a user creates a new order or cancels an existing open order or does some other action that would alter the order status, the ccxt library will remember the entire order info in its cache. Upon a subsequent call to an emulated `fetchOrder`, `fetchOrders` or `fetchClosedOrders` method, the exchange instance will send a single request to **''fetchOpenOrders''** and will compare currently fetched open orders with the orders stored in cache previously. The ccxt library will check each cached order and will try to match it with a corresponding fetched open order. When the cached order isn't present in the open orders fetched from the exchange anymore, the library marks the cached order as `closed` (filled). The call to a `fetchOrder`, `fetchOrders`, `fetchClosedOrders` will then return the updated orders from `.orders` cache to the user.

The same logic can be put shortly: *if a cached order is not found in fetched open orders it isn't open anymore, therefore, closed*. This makes the library capable of tracking the order status and order history even with exchanges that don't have that functionality in their API natively. This is true for all methods that query orders or manipulate (place, cancel or edit) orders in any way.

In most cases the `.orders` cache will work transparently for the user. Most often the exchanges themselves have a sufficient set of methods. However, with some exchanges not having a complete API, the `.orders` cache has the following known limitations:

- If the user does not save the `.orders` cache between program runs and does not restore it upon launching a new run, the `.orders` cache will be lost, for obvious reasons. Therefore upon a call to fetchClosedOrders later on a different run, the exchange instance will return an empty list of orders. Without a properly restored cache a fresh new instance of the exchange won't be able to know anything about the orders that were closed and canceled (no history of orders).
- If the API keypair is shared across multiple exchange instances (e.g. when the user accesses the same exchange account in a multithreaded environment or in simultaneously launched separate scripts). Each particular instance would not be able to know anything about the orders created or canceled by other instances. This means that the order cache is not shared, and, in general, the same API keypair should not be shared across multiple instances accessing the private API. Otherwise it will cause side-effects with nonces and cached data falling out of sync.
- If the order was placed or canceled from outside of ccxt (on the exchange's website or by other means), the new order status won't arrive to the cache and ccxt won't be able to return it properly later.
- If an order's cancelation request bypasses ccxt then the library will not be able to find the order in the list of open orders returned from a subsequent call to `fetchOpenOrders()`. Thus the library will mark the cached order with a `'closed'` status.
- When `fetchOrder(id)` is emulated, the library will not be able to return a specific order, if it was not cached previously or if a change of the order' status was done bypassing ccxt. In that case the library will throw an `OrderNotFound` exception.
- If an unhandled error leads to a crash of the application and the `.orders` cache isn't saved and restored upon restart, the cache will be lost. Handling the exceptions properly is the responsibility of the user. One has to pay **extra care** when implementing proper *error handling*, otherwise the `.orders` cache may fall out of sync.

Note: the order cache functionality is to be reworked soon to obtain the order statuses from private trades history, where available. This is a work in progress, aimed at adding full-featured support for order fees, costs and other info. More about it here: https://github.com/ccxt/ccxt/issues/569.

Purging Cached Orders

With some long-running instances it might be critical to free up used resources when they aren't needed anymore. Because in active trading the `.orders` cache can grow pretty big, the ccxt library offers the `purgeCachedOrders/purge_cached_orders` method for clearing old non-open orders from cache where `(order['timestamp'] < before) && (order['status'] != 'open')` and freeing used memory for other purposes. The purging method accepts one single argument named `before`:

```
// JavaScript

// keep last 24 hours of history in cache
before = exchange.milliseconds () - 24 * 60 * 60 * 1000

// purge all closed and canceled orders "older" or issued "before" that time
exchange.purgeCachedOrders (before)
```

```
# Python

# keep last hour of history in cache
before = exchange.milliseconds () - 1 * 60 * 60 * 1000

# purge all closed and canceled orders "older" or issued "before" that time
exchange.purge_cached_orders (before)
```

```
// PHP

// keep last 24 hours of history in cache
$before = $exchange->milliseconds () - 24 * 60 * 60 * 1000;

// purge all closed and canceled orders "older" or issued "before" that time
$exchange->purge_cached_orders ($before);
```

By Order Id

To get the details of a particular order by its id, use the fetchOrder / fetch_order method. Some exchanges also require a symbol even when fetching a particular order by id.

The signature of the fetchOrder/fetch_order method is as follows:

```
if (exchange.has['fetchOrder']) {
    //  you can use the params argument for custom overrides
    let order = await exchange.fetchOrder (id, symbol = undefined, params = {})
}
```

Some exchanges don't have an endpoint for fetching an order by id, ccxt will emulate it where possible. For now it may still be missing here and there, as this is a work in progress.

You can pass custom overrided key-values in the additional params argument to supply a specific order type, or some other setting if needed.

Below are examples of using the fetchOrder method to get order info from an authenticated exchange instance:

```
// JavaScript
(async function () {
    const order = await exchange.fetchOrder (id)
    console.log (order)
}) ()
```

```
# Python 2/3 (synchronous)
if exchange.has['fetchOrder']:
    order = exchange.fetch_order(id)
    print(order)
```

(continues on next page)

(continued from previous page)

```
# Python 3.5+ asyncio (asynchronous)
import asyncio
import ccxt.async_support as ccxt
if exchange.has['fetchOrder']:
    order = asyncio.get_event_loop().run_until_complete(exchange.fetch_order(id))
    print(order)
```

```
// PHP
if ($exchange->has['fetchOrder']) {
    $order = $exchange->fetch_order ($id);
    var_dump ($order);
}
```

All Orders

```
if (exchange.has['fetchOrders'])
    exchange.fetchOrders (symbol = undefined, since = undefined, limit = undefined,␣
↪params = {})
```

Some exchanges don't have an endpoint for fetching all orders, ccxt will emulate it where possible. For now it may still be missing here and there, as this is a work in progress.

Open Orders

```
if (exchange.has['fetchOpenOrders'])
    exchange.fetchOpenOrders (symbol = undefined, since = undefined, limit =␣
↪undefined, params = {})
```

Closed Orders

Do not confuse *closed orders* with *trades* aka *fills* ! An order can be closed (filled) with multiple opposing trades! So, a *closed order* is not the same as a *trade*. In general, the order does not have a `fee` at all, but each particular user trade does have `fee`, `cost` and other properties. However, many exchanges propagate those properties to the orders as well.

Some exchanges don't have an endpoint for fetching closed orders, ccxt will emulate it where possible. For now it may still be missing here and there, as this is a work in progress.

```
if (exchange.has['fetchClosedOrders'])
    exchange.fetchClosedOrders (symbol = undefined, since = undefined, limit =␣
↪undefined, params = {})
```

11.4.2 Order Structure

Most of methods returning orders within ccxt unified API will usually yield an order structure as described below:

```
{
    'id':                '12345-67890:09876/54321', // string
    'datetime':          '2017-08-17 12:42:48.000', // ISO8601 datetime of 'timestamp
↪' with milliseconds
```

(continues on next page)

(continued from previous page)

```
    'timestamp':          1502962946216, // order placing/opening Unix timestamp in␣
↪milliseconds
    'lastTradeTimestamp': 1502962956216, // Unix timestamp of the most recent trade␣
↪on this order
    'status':     'open',         // 'open', 'closed', 'canceled'
    'symbol':     'ETH/BTC',      // symbol
    'type':       'limit',        // 'market', 'limit'
    'side':       'buy',          // 'buy', 'sell'
    'price':       0.06917684,    // float price in quote currency
    'amount':      1.5,           // ordered amount of base currency
    'filled':      1.1,           // filled amount of base currency
    'remaining':   0.4,           // remaining amount to fill
    'cost':        0.076094524,   // 'filled' * 'price'
    'trades':    [ ... ],         // a list of order trades/executions
    'fee': {                      // fee info, if available
        'currency': 'BTC',        // which currency the fee is (usually quote)
        'cost': 0.0009,           // the fee amount in that currency
        'rate': 0.002,            // the fee rate (if available)
    },
    'info': { ... },              // the original unparsed order structure as is
}
```

- The work on `'fee'` info is still in progress, fee info may be missing partially or entirely, depending on the exchange capabilities.
- The `fee` currency may be different from both traded currencies (for example, an ETH/BTC order with fees in USD).
- The `lastTradeTimestamp` timestamp may have no value and may be `undefined/None/null` where not supported by the exchange or in case of an open order (an order that has not been filled nor partially filled yet).
- The `lastTradeTimestamp`, if any, designates the timestamp of the last trade, in case the order is filled fully or partially, otherwise `lastTradeTimestamp` is `undefined/None/null`.
- Order `status` prevails or has precedence over the `lastTradeTimestamp`.
- The `cost` of an order is: `{ filled * price }`
- The `cost` of an order means the total *quote* volume of the order (whereas the `amount` is the *base* volume). The value of `cost` should be as close to the actual most recent known order cost as possible. The `cost` field itself is there mostly for convenience and can be deduced from other fields.

11.4.3 Placing Orders

To place an order you will need the following information:

- `symbol`, a string literal symbol of the market you wish to trade on, like `BTC/USD`, `ZEC/ETH`, `DOGE/DASH`, etc... Make sure the symbol in question exists with the target exchange and is available for trading.
- `side`, a string literal for the direction of your order, `buy` or `sell`. When you place a buy order you give quote currency and receive base currency. For example, buying `BTC/USD` means that you will receive bitcoins for your dollars. When you are selling `BTC/USD` the outcome is the opposite and you receive dollars for your bitcoins.
- `type`, a string literal type of order, ccxt currently supports `market` and `limit` orders

- `amount`, how much of currency you want to trade. This usually refers to base currency of the trading pair symbol, though some exchanges require the amount in quote currency and a few of them require base or quote amount depending on the side of the order. See their API docs for details.
- `price`, how much quote currency you are willing to pay for a trade lot of base currency (for limit orders only)

A successful call to a unified method for placing market or limit orders returns the following structure:

```
{
    'id': 'string',  // order id
    'info': { ... }, // decoded original JSON response from the exchange as is
}
```

Some exchanges will allow to trade with limit orders only. See their docs for details.

Market Orders

Market price orders are also known as *spot price orders*, *instant orders* or simply *market orders*. A market order gets executed immediately. The matching engine of the exchange closes the order (fulfills it) with one or more transactions from the top of the order book stack.

The exchange will close your market order for the best price available. You are not guaranteed though, that the order will be executed for the price you observe prior to placing your order. There can be a slight change of the price for the traded market while your order is being executed, also known as *price slippage*. The price can slip because of networking roundtrip latency, high loads on the exchange, price volatility and other factors. When placing a market order you don't need to specify the price of the order.

```
// camelCaseNotation
exchange.createMarketBuyOrder (symbol, amount[, params])
exchange.createMarketSellOrder (symbol, amount[, params])

// underscore_notation
exchange.create_market_buy_order (symbol, amount[, params])
exchange.create_market_sell_order (symbol, amount[, params])
```

Note, that some exchanges will not accept market orders (they allow limit orders only). In order to detect programmatically if the exchange in question does support market orders or not, you can use the `.has['createMarketOrder']` exchange property:

```
// JavaScript
if (exchange.has['createMarketOrder']) {
    ...
}
```

```
# Python
if exchange.has['createMarketOrder']:
    ...
```

```
// PHP
if ($exchange->has['createMarketOrder']) {
    ...
}
```

Emulating Market Orders With Limit Orders

It is also possible to emulate a `market` order with a `limit` order.

WARNING this method can be risky due to high volatility, use it at your own risk and only use it when you know really well what you're doing!

Most of the time a `market sell` can be emulated with a `limit sell` at a very low price – the exchange will automatically make it a taker order for market price (the price that is currently in your best interest from the ones that are available in the order book). When the exchange detects that you're selling for a very low price it will automatically offer you the best buyer price available from the order book. That is effectively the same as placing a market sell order. Thus market orders can be emulated with limit orders (where missing).

The opposite is also true – a `market buy` can be emulated with a `limit buy` for a very high price. Most exchanges will again close your order for best available price, that is, the market price.

However, you should never rely on that entirely, **ALWAYS test it with a small amount first!** You can try that in their web interface first to verify the logic. You can sell the minimal amount at a specified limit price (an affordable amount to lose, just in case) and then check the actual filling price in trade history.

Limit Orders

Limit price orders are also known as *limit orders*. Some exchanges accept limit orders only. Limit orders require a price (rate per unit) to be submitted with the order. The exchange will close limit orders if and only if market price reaches the desired level.

```
// camelCaseStyle
exchange.createLimitBuyOrder (symbol, amount, price[, params])
exchange.createLimitSellOrder (symbol, amount, price[, params])

// underscore_style
exchange.create_limit_buy_order (symbol, amount, price[, params])
exchange.create_limit_sell_order (symbol, amount, price[, params])
```

Custom Order Params

Some exchanges allow you to specify optional parameters for your order. You can pass your optional parameters and override your query with an associative array using the `params` argument to your unified API call. All custom params are exchange-specific, of course, and aren't interchangeable, do not expect those custom params for one exchange to work with another exchange.

```
// JavaScript
// use a custom order type
bitfinex.createLimitSellOrder ('BTC/USD', 1, 10, { 'type': 'trailing-stop' })
```

```
# Python
# add a custom order flag
kraken.create_market_buy_order('BTC/USD', 1, {'trading_agreement': 'agree'})
```

```
// PHP
// add custom user id to your order
$hitbtc->create_order ('BTC/USD', 'limit', 'buy', 1, 3000, array ('clientOrderId' =>
↪'123'));
```

11.4.4 Canceling Orders

To cancel an existing order pass the order id to `cancelOrder (id, symbol, params) / cancel_order (id, symbol, params)` method. Note, that some exchanges require a second symbol parameter even to cancel a known order by id. The usage is shown in the following examples:

```
// JavaScript
exchange.cancelOrder ('1234567890') // replace with your order id here (a string)
```

```
# Python
exchange.cancel_order ('1234567890') # replace with your order id here (a string)
```

```
// PHP
$exchange->cancel_order ('1234567890'); // replace with your order id here (a string)
```

Exceptions on order canceling

The `cancelOrder()` is usually used on open orders only. However, it may happen that your order gets executed (filled and closed) before your cancel-request comes in, so a cancel-request might hit an already-closed order.

A cancel-request might also throw a `NetworkError` indicating that the order might or might not have been canceled successfully and whether you need to retry or not. Consecutive calls to `cancelOrder()` may hit an already canceled order as well.

As such, `cancelOrder()` can throw an `OrderNotFound` exception in these cases: - canceling an already-closed order - canceling an already-canceled order

11.5 Personal Trades

```
- this part of the unified API is currenty a work in progress
- there may be some issues and missing implementations here and there
- contributions, pull requests and feedback appreciated
```

11.5.1 How Orders Are Related To Trades

A trade is also often called `a fill`. Each trade is a result of order execution. Note, that orders and trades have a one-to-many relationship: an execution of one order may result in several trades. However, when one order matches another opposing order, the pair of two matching orders yields one trade. Thus, when an order matches multiple opposing orders, this yields multiple trades, one trade per each pair of matched orders.

To put it shortly, an order can contain *one or more* trades. Or, in other words, an order can be *filled* with one or more trades.

For example, an orderbook can have the following orders (whatever trading symbol or pair it is):

```
    | price  | amount
----|----------------
  a |  1.200 | 200
  s |  1.100 | 300
  k |  0.900 | 100
----|----------------
  b |  0.800 | 100
```

(continues on next page)

(continued from previous page)

```
  i |  0.700 | 200
  d |  0.500 | 100
```

All specific numbers above aren't real, this is just to illustrate the way orders and trades are related in general.

A seller decides to place a sell limit order on the ask side for a price of 0.700 and an amount of 150.

```
    | price  | amount
----|--------------  ↓
  a |  1.200 | 200    ↓
  s |  1.100 | 300    ↓
  k |  0.900 | 100    ↓
----|--------------  ↓
  b |  0.800 | 100    ↓ sell 150 for 0.700
  i |  0.700 | 200    --------------------
  d |  0.500 | 100
```

As the price and amount of the incoming sell (ask) order cover more than one bid order (orders `b` and `i`), the following sequence of events usually happens within an exchange engine very quickly, but not immediately:

1. Order `b` is matched against the incoming sell because their prices intersect. Their volumes *"mutually annihilate"* each other, so, the bidder gets 100 for a price of 0.800. The seller (asker) will have his sell order partially filled by bid volume 100 for a price of 0.800. Note that for this filled part of the order the seller gets a better price than he asked for initially (0.8 instead of 0.7). Most conventional exchanges fill orders for the best price available.
2. A trade is generated for the order `b` against the incoming sell order. That trade *"fills"* the entire order `b` and most of the sell order. One trade is generated pear each pair of matched orders, whether the amount was filled completely or partially. In this example the amount of 100 fills order `b` completely (closed the order `b`) and also fills the selling order partially (leaves it open in the orderbook).
3. Order `b` now has a status of `closed` and a filled volume of 100. It contains one trade against the selling order. The selling order has `open` status and a filled volume of 100. It contains one trade against order `b`. Thus each order has just one fill-trade so far.
4. The incoming sell order has a filled amount of 100 and has yet to fill the remaining amount of 50 from its initial amount of 150 in total.
5. Order `i` is matched against the remaining part of incoming sell, because their prices intersect. The amount of buying order `i` which is 200 completely annihilates the remaining sell amount of 50. The order `i` is filled partially by 50, but the rest of its volume, namely the remaining amount of 150 will stay in the orderbook. The selling order, however, is filled completely by this second match.
6. A trade is generated for the order `i` against the incoming sell order. That trade partially fills order `i`. And completes the filling of the sell order. Again, this is just one trade for a pair of matched orders.
7. Order `i` now has a status of `open`, a filled amount of 50, and a remaining amount of 150. It contains one filling trade against the selling order. The selling order has a `closed` status now, as it was completely filled its total initial amount of 150. However, it contains two trades, the first against order `b` and the second against order `i`. Thus each order can have one or more filling trades, depending on how their volumes were matched by the exchange engine.

After the above sequence takes place, the updated orderbook will look like this.

```
    | price  | amount
----|----------------
  a |  1.200 | 200
  s |  1.100 | 300
  k |  0.900 | 100
```

(continues on next page)

(continued from previous page)

```
----|----------------
  i |  0.700 | 150
  d |  0.500 | 100
```

Notice that the order `b` has disappeared, the selling order also isn't there. All closed and fully-filled orders disappear from the orderbook. The order `i` which was filled partially and still has a remaining volume and an `open` status, is still there.

11.5.2 Recent Trades

```
// JavaScript
// fetchMyTrades (symbol = undefined, since = undefined, limit = undefined, params =
↪{})

if (exchange.has['fetchMyTrades']) {
    const trades = await exchange.fetchMyTrades (symbol, since, limit, params)
}
```

```
# Python
# fetch_my_trades (symbol = None, since = None, limit = None, params = {})

if exchange.has['fetchMyTrades']:
    exchange.fetch_my_trades (symbol = None, since = None, limit = None, params = {})
```

```
// PHP
// fetch_my_trades ($symbol = null, $since = null, $limit = null, $params = array ())

if ($exchange->has['fetchMyTrades']) {
    $trades = $exchange->fetch_my_trades ($symbol, $since, $limit, $params);
}
```

Returns ordered array `[]` of trades (most recent trade last).

Trade structure

```
{
    'info':         { ... },                    // the original decoded JSON as is
    'id':           '12345-67890:09876/54321',  // string trade id
    'timestamp':    1502962946216,              // Unix timestamp in milliseconds
    'datetime':     '2017-08-17 12:42:48.000',  // ISO8601 datetime with milliseconds
    'symbol':       'ETH/BTC',                  // symbol
    'order':        '12345-67890:09876/54321',  // string order id or undefined/None/
↪null
    'type':         'limit',                    // order type, 'market', 'limit' or␣
↪undefined/None/null
    'side':         'buy',                      // direction of the trade, 'buy' or
↪'sell'
    'takerOrMaker': 'taker'                     // string, 'taker' or 'maker'
    'price':        0.06917684,                 // float price in quote currency
    'amount':       1.5,                        // amount of base currency
    'cost':         0.10376526,                 // total cost (including fees),␣
↪`price * amount`
    'fee':          {                           // provided by exchange or calculated␣
↪by ccxt
```

(continues on next page)

(continued from previous page)

```
        'cost':  0.0015,                        // float
        'currency': 'ETH',                      // usually base currency for buys,␣
↪quote currency for sells
        'rate': 0.002,                          // the fee rate (if available)
    },
}
```

- The work on `'fee'` info is still in progress, fee info may be missing partially or entirely, depending on the exchange capabilities.
- The `fee` currency may be different from both traded currencies (for example, an ETH/BTC order with fees in USD).
- The `cost` of the trade means `amount * price`. It is the total *quote* volume of the trade (whereas `amount` is the *base* volume). The cost field itself is there mostly for convenience and can be deduced from other fields.

11.5.3 Trades By Order Id

UNDER CONSTRUCTION

11.6 Funding Your Account

```
- this part of the unified API is currenty a work in progress
- there may be some issues and missing implementations here and there
- contributions, pull requests and feedback appreciated
```

11.6.1 Deposit

```
fetchDepositAddress (code, params = {})
createDepositAddress (code, params = {})
```

- `code` is the currency code (uppercase string)
- `params` contains optional extra overrides

```
{
    'currency': currency, // currency code
    'address': address,   // address in terms of requested currency
    'tag': tag,           // tag / memo / paymentId for particular currencies (XRP,␣
↪XMR, ...)
    'info': response,     // raw unparsed data as returned from the exchange
}
```

With certain currencies, like AEON, BTS, GXS, NXT, SBD, STEEM, STR, XEM, XLM, XMR, XRP, an additional argument `tag` is usually required by exchanges. Other currencies will have the `tag` set to `undefined / None / null`. The tag is a memo or a message or a payment id that is attached to a withdrawal transaction. The tag is mandatory for those currencies and it identifies the recipient user account.

Be careful when specifying the `tag` and the `address`. The `tag` is **NOT an arbitrary user-defined string** of your choice! You cannot send user messages and comments in the `tag`. The purpose of the `tag` field is to address your wallet properly, so it must be correct. You should only use the `tag` received from the exchange you're working with, otherwise your transaction might never arrive to its destination.

11.6.2 Withdraw

```
// JavaScript
exchange.withdraw (code, amount, address, tag = undefined, params = {})
```

```
# Python
exchange.withdraw(code, amount, address, tag=None, params={})
```

```
// PHP
$exchange->withdraw ($code, $amount, $address, $tag = null, $params = array ())
```

The `code` is the currency code (usually three or more uppercase letters, but can be different in some cases).

The withdraw method returns a dictionary containing the withdrawal id, which is usually the txid of the onchain transaction itself, or an internal *withdrawal request id* registered within the exchange. The returned value looks as follows:

```
{
    'info' { ... },      // unparsed reply from the exchange, as is
    'id': '12345567890', // string withdrawal id, if any
}
```

Some exchanges require a manual approval of each withdrawal by means of 2FA (2-factor authentication). In order to approve your withdrawal you usually have to either click their secret link in your email inbox or enter a Google Authenticator code or an Authy code on their website to verify that withdrawal transaction was requested intentionally.

In some cases you can also use the withdrawal id to check withdrawal status later (whether it succeeded or not) and to submit 2FA confirmation codes, where this is supported by the exchange. See their docs for details.

11.6.3 Transactions

Transaction Structure

```
{
    'info':      { ... },    // the json response from the exchange, as is
    'id':       '123456',    // exchange-specific transaction id, string
    'txid':     '0x68bfb29821c50ca35ef3762f887fd3211e4405aba1a94e448a4f218b850358f0',
    'timestamp': 1534081184515,            // timestamp in milliseconds
    'datetime': '2018-08-12T13:39:44.515Z', // ISO8601 string of the timestamp
    'address':  '0x02b0a9b7b4cDe774af0f8e47cb4f1c2ccdEa0806', // "from" or "to"
    'type':     'deposit',   // or 'withdrawal', string
    'amount':    1.2345,     // float
    'currency': 'ETH',       // a common unified currency code, string
    'status':   'pending',   // 'ok', 'failed', 'canceled', string
    'updated':   undefined,  // UTC timestamp in ms of most recent status change
    'fee': {                 // the entire fee structure may be undefined
        'cost': 0.1234,      // float
        'rate': undefined,   // approximately, fee['cost'] / amount, float
    },
}
```

Notes On Transaction Structure

- The `updated` field is the UTC timestamp in milliseconds of the most recent change of status of that funding operation, be it `withdrawal` or `deposit`. It is necessary if you want to track your changes in time, beyond a static snapshot. For example, if the exchange in question reports `created_at` and `confirmed_at` for a transaction, then the `updated` field will take the value of `Math.max (created_at, confirmed_at)`, that is, the timestamp of the most recent change of the status.
- The `updated` field may be undefined in certain exchange-specific cases.
- The `fee` substructure may be missing, if not supplied within the reply coming from the exchange.

Deposits

```
// JavaScript
// fetchDeposits (code = undefined, since = undefined, limit = undefined, params = {})

if (exchange.has['fetchDeposits']) {
    const deposits = await exchange.fetchDeposits (code, since, limit, params)
}
```

```
# Python
# fetch_deposits(code = None, since = None, limit = None, params = {})

if (exchange.has['fetchDeposits']) {
    deposits = exchange.fetch_deposits(code, since, limit, params)
}
```

```
// PHP
// fetch_deposits ($code = null, $since = null, $limit = null, $params = {})

if ($exchange->has['fetchDeposits']) {
    $deposits = $exchange->fetch_deposits ($code, $since, $limit, $params);
}
```

Withdrawals

```
// JavaScript
// fetchWithdrawals (code = undefined, since = undefined, limit = undefined, params =
↪{})

if (exchange.has['fetchWithdrawals']) {
    const withdrawals = await exchange.fetchWithdrawals (code, since, limit, params)
}
```

```
# Python
# fetch_withdrawals(code = None, since = None, limit = None, params = {})

if (exchange.has['fetchWithdrawals']) {
    withdrawals = exchange.fetch_withdrawals(code, since, limit, params)
}
```

```
// PHP
// fetch_withdrawals ($code = null, $since = null, $limit = null, $params = {})

if ($exchange->has['fetchWithdrawals']) {
    $withdrawals = $exchange->fetch_withdrawals ($code, $since, $limit, $params);
}
```

All Transactions

```
// JavaScript
// fetchTransactions (code = undefined, since = undefined, limit = undefined, params␣
↪= {})

if (exchange.has['fetchTransactions']) {
    const transactions = await exchange.fetchTransactions (code, since, limit, params)
}
```

```
# Python
# fetch_transactions(code = None, since = None, limit = None, params = {})

if (exchange.has['fetchTransactions']) {
    transactions = exchange.fetch_transactions(code, since, limit, params)
}
```

```
// PHP
// fetch_transactions ($code = null, $since = null, $limit = null, $params = {})

if ($exchange->has['fetchTransactions']) {
    $transactions = $exchange->fetch_transactions ($code, $since, $limit, $params);
}
```

11.7 Fees

This section of the Unified CCXT API is under development.

Fees are often grouped into two categories:

- Trading fees. Trading fee is the amount payable to the exchange, usually a percentage of volume traded (filled)).
- Funding fees. The amount payable to the exchange upon depositing and withdrawing as well as the underlying crypto transaction fees (tx fees).

Because the fee structure can depend on the actual volume of currencies traded by the user, the fees can be account-specific. Methods to work with account-specific fees:

```
fetchFees (params = {})
fetchTradingFees (params = {})
fetchFundingFees (params = {})
```

The fee methods will return a unified fee structure, which is often present with orders and trades as well. The fee structure is a common format for representing the fee info throughout the library. Fee structures are usually indexed by market or currency.

Because this is still a work in progress, some or all of methods and info described in this section may be missing with this or that exchange.

DO NOT use the ‘‘.fees‘‘ property as most often it contains the predefined/hardcoded info, which is now deprecated. Actual fees should only be accessed from markets and currencies.

11.7.1 Fee structure

```
{
    'type': takerOrMaker,
    'currency': 'BTC', // the unified fee currency code
    'rate': percentage, // the fee rate, 0.05% = 0.0005, 1% = 0.01, ...
    'cost': feePaid, // the fee cost (amount * fee rate)
}
```

11.7.2 Trading Fees

Trading fees are properties of markets. Most often trading fees are loaded into the markets by the `fetchMarkets` call. Sometimes, however, the exchanges serve fees from different endpoints.

The `calculateFee` method can be used to precalculate trading fees that will be paid. **WARNING! This method is experimental, unstable and may produce incorrect results in certain cases**. You should only use it with caution. Actual fees may be different from the values returned from `calculateFee`, this is just for precalculation. Do not rely on precalculated values, because market conditions change frequently. It is difficult to know in advance whether your order will be a market taker or maker.

```
calculateFee (symbol, type, side, amount, price, takerOrMaker = 'taker', params = {})
```

The `calculateFee` method will return a unified fee structure with precalculated fees for an order with specified params.

Accessing trading fee rates should be done via the `.markets` property, like so:

```
exchange.markets['ETH/BTC']['taker'] // taker fee rate for ETH/BTC
exchange.markets['BTC/USD']['maker'] // maker fee rate for BTC/USD
```

Maker fees are paid when you provide liquidity to the exchange i.e. you *market-make* an order and someone else fills it. Maker fees are usually lower than taker fees. Similarly, taker fees are paid when you *take* liquidity from the exchange and fill someone else’s order.

11.7.3 Funding Fees

Funding fees are properties of currencies (account balance).

Accessing funding fee rates should be done via the `.currencies` property. This aspect is not unified yet and is subject to change.

```
exchange.currencies['ETH']['fee'] // tx/withdrawal fee rate for ETH
exchange.currencies['BTC']['fee'] // tx/withdrawal fee rate for BTC
```

11.7.4 Ledger

UNDER CONSTRUCTION

11.8 Overriding The Nonce

The default nonce is a 32-bit Unix Timestamp in seconds. You should override it with a milliseconds-nonce if you want to make private requests more frequently than once per second! Most exchanges will throttle your requests if you hit their rate limits, readAPI docs for your exchange**carefully!**

In case you need to reset the nonce it is much easier to create another pair of keys for using with private APIs. Creating new keys and setting up a fresh unused keypair in your config is usually enough for that.

In some cases you are unable to create new keys due to lack of permissions or whatever. If that happens you can still override the nonce. Base market class has the following methods for convenience:

- `seconds ()`: returns a Unix Timestamp in seconds.
- `milliseconds ()`: same in milliseconds (ms = 1000 * s, thousandths of a second).
- `microseconds ()`: same in microseconds (μs = 1000 * ms, millionths of a second).

There are exchanges that confuse milliseconds with microseconds in their API docs, let's all forgive them for that, folks. You can use methods listed above to override the nonce value. If you need to use the same keypair from multiple instances simultaneously use closures or a common function to avoid nonce conflicts. In Javascript you can override the nonce by providing a `nonce` parameter to the exchange constructor or by setting it explicitly on exchange object:

```
// JavaScript

// A: custom nonce redefined in constructor parameters
let nonce = 1
let kraken1 = new ccxt.kraken ({ nonce: () => nonce++ })

// B: nonce redefined explicitly
let kraken2 = new ccxt.kraken ()
kraken2.nonce = function () { return nonce++ } // uses same nonce as kraken1

// C: milliseconds nonce
let kraken3 = new ccxt.kraken ({
    nonce: function () { return this.milliseconds () },
})

// D: newer ES syntax
let kraken4 = new ccxt.kraken ({
    nonce () { return this.milliseconds () },
})
```

In Python and PHP you can do the same by subclassing and overriding nonce function of a particular exchange class:

```
# Python

# A: the shortest
gdax = ccxt.gdax({'nonce': ccxt.Exchange.milliseconds})

# B: custom nonce
class MyKraken(ccxt.kraken):
    n = 1
    def nonce(self):
        return self.n += 1

# C: milliseconds nonce
class MyBitfinex(ccxt.bitfinex):
```

(continues on next page)

(continued from previous page)

```
    def nonce(self):
        return self.milliseconds()

# D: milliseconds nonce inline
hitbtc = ccxt.hitbtc({
    'nonce': lambda: int(time.time() * 1000)
})

# E: milliseconds nonce
acx = ccxt.acx({'nonce': lambda: ccxt.Exchange.milliseconds()})
```

```
// PHP

// A: custom nonce value
class MyOKCoinUSD extends \ccxt\okcoinusd {
    public function __construct ($options = array ()) {
        parent::__construct (array_merge (array ('i' => 1), $options));
    }
    public function nonce () {
        return $this->i++;
    }
}

// B: milliseconds nonce
class MyZaif extends \ccxt\zaif {
    public function __construct ($options = array ()) {
        parent::__construct (array_merge (array ('i' => 1), $options));
    }
    public function nonce () {
        return $this->milliseconds ();
    }
}
```

CHAPTER 12

Error Handling

All exceptions are derived from the base BaseError exception, which, in its turn, is defined in the ccxt library like so:

```
// JavaScript
class BaseError extends Error {
    constructor () {
        super ()
        // a workaround to make `instanceof BaseError` work in ES5
        this.constructor = BaseError
        this.__proto__   = BaseError.prototype
    }
}
```

```
# Python
class BaseError (Exception):
    pass
```

```
// PHP
class BaseError extends \Exception {}
```

Below is an outline of exception inheritance hierarchy:

```
+ BaseError
|
+---+ ExchangeError
|   |
|   +---+ NotSupported
|   |
|   +---+ AuthenticationError
|   |   |
|   |   +---+ PermissionDenied
|   |
|   +---+ InsufficientFunds
|   |
```

(continues on next page)

(continued from previous page)

```
|    +---+ InvalidAddress
|    |
|    +---+ InvalidOrder
|        |
|        +---+ OrderNotFound
|
+---+ NetworkError (recoverable)
    |
    +---+ DDoSProtection
    |
    +---+ RequestTimeout
    |
    +---+ ExchangeNotAvailable
    |
    +---+ InvalidNonce
```

The `BaseError` class is a generic error class for all sorts of errors, including accessibility and request/response mismatch. Users should catch this exception at the very least, if no error differentiation is required.

12.1 ExchangeError

This exception is thrown when an exchange server replies with an error in JSON. Possible reasons:

- endpoint is switched off by the exchange
- symbol not found on the exchange
- required parameter is missing
- the format of parameters is incorrect
- an exchange replies with an unclear answer

Other exceptions derived from `ExchangeError`:

- `NotSupported`: This exception is raised if the endpoint is not offered/not supported by the exchange API.
- `AuthenticationError`: Raised when an exchange requires one of the API credentials that you've missed to specify, or when there's a mistake in the keypair or an outdated nonce. Most of the time you need `apiKey` and `secret`, sometimes you also need `uid` and/or `password`.
- `PermissionDenied`: Raised when there's no access for specified action or insufficient permissions on the specified `apiKey`.
- `InsufficientFunds`: This exception is raised when you don't have enough currency on your account balance to place an order.
- `InvalidAddress`: This exception is raised upon encountering a bad funding address or a funding address shorter than `.minFundingAddressLength` (10 characters by default) in a call to `fetchDepositAddress`, `createDepositAddress` or `withdraw`.
- `InvalidOrder`: This exception is the base class for all exceptions related to the unified order API.
- `OrderNotFound`: Raised when you are trying to fetch or cancel a non-existent order.

12.2 NetworkError

All errors related to networking are usually recoverable, meaning that networking problems, traffic congestion, unavailability is usually time-dependent. Making a retry later is usually enough to recover from a NetworkError, but if it doesn't go away, then it may indicate some persistent problem with the exchange or with your connection.

12.2.1 DDoSProtection

This exception is thrown in either of two cases:

- when Cloudflare or Incapsula rate limiter restrictions are enforced per user or region/location
- when the exchange restricts user access for requesting the endpoints in question too frequently

In addition to default error handling, the ccxt library does a case-insensitive search in the response received from the exchange for one of the following keywords:

- `cloudflare`
- `incapsula`
- `overload`
- `ddos`

12.2.2 RequestTimeout

This exception is raised when the connection with the exchange fails or data is not fully received in a specified amount of time. This is controlled by the `timeout` option. When a `RequestTimeout` is raised, the user doesn't know the outcome of a request (whether it was accepted by the exchange server or not).

Thus it's advised to handle this type of exception in the following manner:

- for fetching requests it is safe to retry the call
- for a request to `cancelOrder` a user is required to retry the same call the second time. If instead of a retry a user calls a `fetchOrder`, `fetchOrders`, `fetchOpenOrders` or `fetchClosedOrders` right away without a retry to call `cancelOrder`, this may cause the `` `.orders `` cache <#orders-cache>‘__ to fall out of sync. A subsequent retry to `cancelOrder` will return one of the following possible results:
  - a request is completed successfully, meaning the order has been properly canceled now
  - an `OrderNotFound` exception is raised, which means the order was either already canceled on the first attempt or has been executed (filled and closed) in the meantime between the two attempts. Note, that the order will still have an `'open'` status in the `.orders` cache. To determine the actual order status you'll need to call `fetchOrder` to update the cache properly (where available from the exchange). If the `fetchOrder` method is `'emulated'` the ccxt library will mark the order as `'closed'`. The user has to call `fetchBalance` and set the order status to `'canceled'` manually if the balance hasn't changed (a trade didn't not occur).
- if a request to `createOrder` fails with a `RequestTimeout` the user should:
  - update the `.orders` cache with a call to `fetchOrders`, `fetchOpenOrders`, `fetchClosedOrders` to check if the request to place the order has succeeded and the order is now open
  - if the order is not `'open'` the user should `fetchBalance` to check if the balance has changed since the order was created on the first run and then was filled and closed by the time of the second check. Note

that `fetchBalance` relies on the `.orders` cache for *balance inference* and thus should only be called after updating the cache!

12.2.3 ExchangeNotAvailable

The ccxt library throws this error if it detects any of the following keywords in response:

- `offline`
- `unavailable`
- `busy`
- `retry`
- `wait`
- `maintain`
- `maintenance`
- `maintenancing`

12.2.4 InvalidNonce

Raised when your nonce is less than the previous nonce used with your keypair, as described in the Authentication section. This type of exception is thrown in these cases (in order of precedence for checking):

- You are not rate-limiting your requests or sending too many of them too often.
- Your API keys are not fresh and new (have been used with some different software or script already, just always create a new keypair when you add this or that exchange).
- The same keypair is shared across multiple instances of the exchange class (for example, in a multithreaded environment or in separate processes).
- Your system clock is out of synch. System time should be synched with UTC in a non-DST timezone at a rate of once every ten minutes or even more frequently because of the clock drifting. **Enabling time synch in Windows is usually not enough!** You have to set it up with the OS Registry (Google *"time synch frequency"* for your OS).

CHAPTER 13

Troubleshooting

In case you experience any difficulty connecting to a particular exchange, do the following in order of precedence:

- Make sure that you have the most recent version of ccxt.
- Check the CHANGELOG for recent updates.
- Turn `verbose = true` to get more detail about it.
- Python people can turn on DEBUG logging level with a standard pythonic logger, by adding these two lines to the beginning of their code:

```
import logging
logging.basicConfig(level=logging.DEBUG)
```

- Check your API credentials. Try a fresh new keypair if possible.
- If it is a Cloudflare protection error, try these examples:
  - https://github.com/ccxt/ccxt/blob/master/examples/js/bypass-cloudflare.js
  - https://github.com/ccxt/ccxt/blob/master/examples/py/bypass-cloudflare.py
  - https://github.com/ccxt/ccxt/blob/master/examples/py/bypass-cloudflare-with-cookies.py
- Check your nonce. If you used your API keys with other software, you most likely should *override your nonce function* to match your previous nonce value. A nonce usually can be easily reset by generating a new unused keypair. If you are getting nonce errors with an existing key, try with a new API key that hasn't been used yet.
- Check your request rate if you are getting nonce errors. Your private requests should not follow one another quickly. You should not send them one after another in a split second or in short time. The exchange will most likely ban you if you don't make a delay before sending each new request. In other words, you should not hit their rate limit by sending unlimited private requests too frequently. Add a delay to your subsequent requests or enable the built-in rate-limiter, like shown in the long-poller examples, also here.
- Read the docs for your exchange and compare your verbose output to the docs.
- Check your connectivity with the exchange by accessing it with your browser.
- Check your connection with the exchange through a proxy. Read the Proxy section for more details.

- Try accesing the exchange from a different computer or a remote server, to see if this is a local or global issue with the exchange.
- Check if there were any news from the exchange recently regarding downtime for maintenance. Some exchanges go offline for updates regularly (like once a week).

13.1 Notes

- Use the `verbose = true` option or instantiate your troublesome exchange with `new ccxt.exchange ({ 'verbose': true })` to see the HTTP requests and responses in details. The verbose output will also be of use for us to debug it if you submit an issue on GitHub.
- Use DEBUG logging in Python!
- As written above, some exchanges are not available in certain countries. You should use a proxy or get a server somewhere closer to the exchange.
- If you are getting authentication errors or *'invalid keys'* errors, those are most likely due to a nonce issue.
- Some exchanges do not state it clearly if they fail to authenticate your request. In those circumstances they might respond with an exotic error code, like HTTP 502 Bad Gateway Error or something that's even less related to the actual cause of the error.
- ...

UNDER CONSTRUCTION

CHAPTER 14

Frequently Asked Questions

14.1 I'm trying to run the code, but it's not working, how do I fix it?

If your question is formulated in a short manner like the above, we won't help. We don't teach programming. If you're unable to read and understand the Manual or you can't follow precisely the guides from the CONTRIBUTING doc on how to report an issue, we won't help either. Read the Manual. You should not risk anyone's money and time without reading the entire Manual very carefully. You should not risk anything if you're not used to a lot of reading with tons of details. Also, if you don't have the confidence with the programming language you're using, there are much better places for coding fundamentals and practice. Search for `python tutorials`, `js videos`, play with examples, this is how other people climb up the learning curve. No shortcuts, if you want to learn something.

When asking a question: - Use the search button for duplicates first! - **Post your request and response in ‘‘verbose‘‘ mode!** It's written and mentioned everywhere, in the Troubleshooting section, in the README and in many answers to similar questions among previous issues and pull requests. - **Post your code** to reproduce the problem. Make it a complete short runnable program, don't swallow the lines and make it as compact as you can (5-10 lines of code), including the instantation code. - **Surround code and output with triple backticks: ‘‘‘GOOD‘‘‘**. - Don't confuse the backtick symbol (‘) with the quote symbol (’): ’‘’BAD”’ - Don't confuse a single backtick with triple backticks: ‘BAD‘ - Post your version number of ccxt - Post your language version number, how do you think we can guess it otherwise?

14.2 I am calling a method and I get an error, what am I doing wrong?

You're not reporting the issue properly) Please, help the community to help you) Read this and follow the steps: https://github.com/ccxt/ccxt/blob/master/CONTRIBUTING.md#how-to-submit-an-issue. Once again, your code to reproduce the issue and your verbose request and response **ARE REQUIRED**. *Just the error traceback, or just the response, or just the request, or just the code – is not enough!*

14.3 I got an incorrect result from a method call, can you help?

Basically the same answer as the previous question. Read and follow **precisely**: https://github.com/ccxt/ccxt/blob/master/CONTRIBUTING.md#how-to-submit-an-issue. Once again, your code to reproduce the issue and your verbose request and response **ARE REQUIRED**. *Just the error traceback, or just the response, or just the request, or just the code – is not enough!*

14.4 Can you implement feature `foo` in exchange `bar`?

Yes, we can. And we will, if nobody else does that before us. There's very little point in asking this type of questions, because the answer is always positive. When someone asks if we can do this or that, the question is not about our abilities, it all boils down to time and management needed for implementing all accumulated feature requests. Moreover, this is an open-source library which is a work in progress. This means, that this project is intended to be developed by the community of users, who are using it. All contributions are welcome! What you're asking is not whether we can or cannot implement it, in fact you're actually telling us to go do that particular task and this is not how we see a voluntary collaboration.

14.5 When will you add feature `foo` for exchange `bar` ? What's the estimated time? When should we expect this?

We don't give promises or estimates on the open-source work. The reasoning behind this is explained in the previous paragraph.

14.6 What's your progress on adding the feature `foo` that was requested earlier? How do you do implementing exchange `bar`?

This type of questions is usually a waste of time, because answering it usually requires too much time for context-switching, and it often takes more time to answer this question, than to actually satisfy the request with code for a new feature or a new exchange. The progress of this open-source project is also open, so, whenever you're wondering how it is doing, take a look into commit history.

14.7 Hey! The fix you've uploaded is in JS, would you fix Python / PHP as well, please?

Our build system generates exchange-specific Python and PHP code for us automatically, so it is transpiled from JS, and there's no need to fix all languages separately one by one. Thus, if it is fixed in JS, it is fixed in Python pip and PHP Composer as well. Just upgrade your version with `pip` or `composer` and you'll be fine. More about it here: https://github.com/ccxt/ccxt/blob/master/CONTRIBUTING.md#multilanguage-support

CHAPTER 15

CCXT – CryptoCurrency eXchange Trading Library

Build Status npm PyPI NPM Downloads Gitter Supported Exchanges Open Collective Twitter Follow

A JavaScript / Python / PHP library for cryptocurrency trading and e-commerce with support for many bitcoin/ether/altcoin exchange markets and merchant APIs.

15.1 Install · Usage · Manual · FAQ · Examples · Contributing · Social

The **CCXT** library is used to connect and trade with cryptocurrency / altcoin exchanges and payment processing services worldwide. It provides quick access to market data for storage, analysis, visualization, indicator development, algorithmic trading, strategy backtesting, bot programming, webshop integration and related software engineering.

It is intended to be used by **coders, developers, technically-skilled traders, data-scientists and financial analysts** for building trading algorithms on top of it.

Current feature list:

- support for many exchange markets, even more upcoming soon
- fully implemented public and private APIs for all exchanges
- all currencies, altcoins and symbols, prices, order books, trades, tickers, etc. . .
- optional normalized data for cross-exchange or cross-currency analytics and arbitrage
- an out-of-the box unified all-in-one API extremely easy to integrate
- works in Node 7.6+, Python 2 and 3, PHP 5.3+, web browsers

15.1.1 Sponsored Promotion

TheOcean

Learn more about The Ocean in CCXT v1.17 Release Announcement.

15.1.2 Certified Cryptocurrency Exchanges

	id	name	certified	ver	doc	countries
BINANCE	binance	Binance	CCXT Certified	*	API	Japan
BITFINEX	bitfinex	Bitfinex	CCXT Certified	1	API	British Virgin Islands
BITTREX	bittrex	Bittrex	CCXT Certified	1.1	API	US
kraken	kraken	Kraken	CCXT Certified	0	API	US
The Ocean	theocean	The Ocean	CCXT Certified	0	API	US

15.1.3 Supported Cryptocurrency Exchange Markets

The ccxt library currently supports the following 131 cryptocurrency exchange markets and trading APIs:

	id	name	certified	ver	doc	countries
1 BROKER	_1broker	1Broker		2	API	US
1BTCXE	_1btcxe	1BTCXE		*	API	Panama
ACX	acx	ACX		2	API	Australia
ALLCOIN	allcoin	Allcoin		1	API	Canada
ANXPRO	anxpro	ANXPro		2	API	Japan, Singapore, Hong Kon
ANYBITS	anybits	Anybits		*	API	Ireland
BCEX	bcex	BCEX		1	API	China, Canada
Bibox	bibox	Bibox		1	API	China, US, South Korea
BigONE	bigone	BigONE		2	API	UK
BINANCE	binance	Binance	CCXT Certified	*	API	Japan
Bit2C	bit2c	Bit2C		*	API	Israel
bitbank	bitbank	bitbank		1	API	Japan

Continue

Table 1 – continued from previous page

	id	name	certified	ver	doc	countries
	bitbay	BitBay		*	API	Malta, EU
	bitfinex	Bitfinex	CCXT Certified	1	API	British Virgin Islands
	bitfinex2	Bitfinex v2		2	API	British Virgin Islands
	bitflyer	bitFlyer		1	API	Japan
	bitforex	Bitforex		1	API	China
	bithumb	Bithumb		*	API	South Korea
	bitkk	bitkk		1	API	China
	bitlish	Bitlish		1	API	UK, EU, Russia
	bitmarket	BitMarket		*	API	Poland, EU
	bitmex	BitMEX		1	API	Seychelles
	bitsane	Bitsane		*	API	Ireland
	bitso	Bitso		3	API	Mexico
	bitstamp	Bitstamp		2	API	UK
	bitstamp1	Bitstamp v1		1	API	UK
	bittrex	Bittrex	CCXT Certified	1.1	API	US
	bitz	Bit-Z		2	API	Hong Kong
	bl3p	BL3P		1	API	Netherlands, EU
	bleutrade	Bleutrade		2	API	Brazil
	braziliex	Braziliex		*	API	Brazil
	btcalpha	BTC-Alpha		1	API	US
	btcbox	BtcBox		1	API	Japan

Continue

Table 1 – continued from previous page

	id	name	certified	ver	doc	countries
	btcchina	BTCChina		1	API	China
	btcexchange	BTCExchange		*	API	Philippines
	btcmarkets	BTC Markets		*	API	Australia
	btctradeim	BtcTrade.im		*	API	Hong Kong
	btctradeua	BTC Trade UA		*	API	Ukraine
	btcturk	BTCTurk		*	API	Turkey
	btcx	BTCX		1	API	Iceland, US, EU
	bxinth	BX.in.th		*	API	Thailand
	ccex	C-CEX		*	API	Germany, EU
	cex	CEX.IO		*	API	UK, EU, Cyprus, Russia
	chbtc	CHBTC		1	API	China
	chilebit	ChileBit		1	API	Chile
	cobinhood	COBINHOOD		1	API	Taiwan
	coinbase	coinbase		2	API	US
	coinbaseprime	Coinbase Prime		*	API	US
	coinbasepro	Coinbase Pro		*	API	US
	coincheck	coincheck		*	API	Japan, Indonesia
	coinegg	CoinEgg		*	API	China, UK
	coinex	CoinEx		1	API	China
	coinexchange	CoinExchange		*	API	India, Japan, South Korea, V
	coinfalcon	CoinFalcon		1	API	UK

Continue

Table 1 – continued from previous page

	id	name	certified	ver	doc	countries
	coinfloor	coinfloor		*	API	UK
	coingi	Coingi		*	API	Panama, Bulgaria, China, US
	coinmarketcap	CoinMarketCap		1	API	US
	coinmate	CoinMate		*	API	UK, Czech Republic, EU
	coinnest	coinnest		*	API	South Korea
	coinone	CoinOne		2	API	South Korea
	coinsecure	Coinsecure		1	API	India
	coinspot	CoinSpot		*	API	Australia
	cointiger	CoinTiger		1	API	China
	coolcoin	CoolCoin		*	API	Hong Kong
	crypton	Crypton		1	API	EU
	cryptopia	Cryptopia		*	API	New Zealand
	deribit	Deribit		1	API	Netherlands
	dsx	DSX		3	API	UK
	ethfinex	Ethfinex		1	API	British Virgin Islands
	exmo	EXMO		1	API	Spain, Russia
	exx	EXX		*	API	China
	fcoin	FCoin		2	API	China
	flowbtc	flowBTC		1	API	Brazil
	foxbit	FoxBit		1	API	Brazil
	fybse	FYB-SE		*	API	Sweden

Continued

Table 1 – continued from previous page

	id	name	certified	ver	doc	countries
	fybsg	FYB-SG		*	API	Singapore
	gatecoin	Gatecoin		*	API	Hong Kong
	gateio	Gate.io		2	API	China
	gdax	GDAX		*	API	US
	gemini	Gemini		1	API	US
	getbtc	GetBTC		*	API	St. Vincent & Grenadines, R
	hadax	HADAX		1	API	China
	hitbtc	HitBTC		1	API	Hong Kong
	hitbtc2	HitBTC v2		2	API	Hong Kong
	huobi	Huobi		3	API	China
	huobicny	Huobi CNY		1	API	China
	huobipro	Huobi Pro		1	API	China
	ice3x	ICE3X		1	API	South Africa
	independentreserve	Independent Reserve		*	API	Australia, New Zealand
	indodax	INDODAX		1.8	API	Indonesia
	itbit	itBit		1	API	US
	jubi	jubi.com		1	API	China
	kraken	Kraken	CCXT Certified	0	API	US
	kucoin	Kucoin		1	API	Hong Kong
	kuna	Kuna		2	API	Ukraine
	lakebtc	LakeBTC		2	API	US

Continued

Table 1 – continued from previous page

	id	name	certified	ver	doc	countries
	lbank	LBank		1	API	China
	liqui	Liqui		3	API	Ukraine
	livecoin	LiveCoin		*	API	US, UK, Russia
	luno	luno		1	API	UK, Singapore, South Africa
	lykke	Lykke		1	API	Switzerland
	mercado	Mercado Bitcoin		3	API	Brazil
	mixcoins	MixCoins		1	API	UK, Hong Kong
	negociecoins	NegocieCoins		3	API	Brazil
	nova	Novaexchange		2	API	Tanzania
	okcoincny	OKCoin CNY		1	API	China
	okcoinusd	OKCoin USD		1	API	China, US
	okex	OKEX		1	API	China, US
	paymium	Paymium		1	API	France, EU
	poloniex	Poloniex		*	API	US
	qryptos	QRYPTOS		2	API	China, Taiwan
	quadrigacx	QuadrigaCX		2	API	Canada
	quoinex	QUOINEX		2	API	Japan, Singapore, Vietnam
	rightbtc	RightBTC		*	API	United Arab Emirates
	southxchange	SouthXchange		*	API	Argentina
	surbitcoin	SurBitcoin		1	API	Venezuela
	theocean	The Ocean	CCXT Certified	0	API	US

Continued

Table 1 – continued from previous page

	id	name	certified	ver	doc	countries
	therock	TheRockTrading		1	API	Malta
	tidebit	TideBit		2	API	Hong Kong
	tidex	Tidex		3	API	UK
	uex	UEX		1.0.3	API	Singapore, US
	urdubit	UrduBit		1	API	Pakistan
	vaultoro	Vaultoro		1	API	Switzerland
	vbtc	VBTC		1	API	Vietnam
	virwox	VirWoX		*	API	Austria, EU
	wex	WEX		3	API	New Zealand
	xbtce	xBTCe		1	API	Russia
	yobit	YoBit		3	API	Russia
	yunbi	YUNBI		2	API	China
	zaif	Zaif		1	API	Japan
	zb	ZB		1	API	China

The list above is updated frequently, new crypto markets, altcoin exchanges, bug fixes, API endpoints are introduced and added on a regular basis. See the Manual for details. If you don't find a cryptocurrency exchange market in the list above and/or want another exchange to be added, post or send us a link to it by opening an issue here on GitHub or via email.

The library is under MIT license, that means it's absolutely free for any developer to build commercial and opensource software on top of it, but use it at your own risk with no warranties, as is.

15.1.4 Install

The easiest way to install the ccxt library is to use builtin package managers:

- ccxt in NPM (JavaScript / Node v7.6+)
- ccxt in PyPI (Python 2 and 3.5.3+)
- ccxt in Packagist/Composer (PHP 5.3+)

This library is shipped as an all-in-one module implementation with minimalistic dependencies and requirements:

- `js/ <https://github.com/ccxt/ccxt/blob/master/js/>`__ in JavaScript
- `python/ <https://github.com/ccxt/ccxt/blob/master/python/>`__ in Python (generated from JS)
- `php/ <https://github.com/ccxt/ccxt/blob/master/php/>`__ in PHP (generated from JS)

You can also clone it into your project directory from ccxt GitHub repository:

```
git clone https://github.com/ccxt/ccxt.git
```

An alternative way of installing this library into your code is to copy a single file manually into your working directory with language extension appropriate for your environment.

15.2 JavaScript (NPM)

JavaScript version of CCXT works both in Node and web browsers. Requires ES6 and `async/await` syntax support (Node 7.6.0+). When compiling with Webpack and Babel, make sure it is not excluded in your `babel-loader` config.

ccxt in NPM

```
npm install ccxt
```

```
var ccxt = require ('ccxt')

console.log (ccxt.exchanges) // print all available exchanges
```

15.3 JavaScript (for use with the <script> tag):

All-in-one browser bundle (dependencies included), served from unpkg CDN, which is a fast, global content delivery network for everything on NPM.

```
<script type="text/javascript" src="https://unpkg.com/ccxt"></script>
```

Creates a global `ccxt` object:

```
console.log (ccxt.exchanges) // print all available exchanges
```

15.4 Python

ccxt in PyPI

```
pip install ccxt
```

```
import ccxt
print(ccxt.exchanges) # print a list of all available exchange classes
```

The library supports concurrent asynchronous mode with asyncio and async/await in Python 3.5.3+

```
import ccxt.async_support as ccxt # link against the asynchronous version of ccxt
```

15.5 PHP

ccxt in PHP with Packagist/Composer (PHP 5.3+)

It requires common PHP modules:

- cURL
- mbstring (using UTF-8 is highly recommended)
- PCRE
- iconv
- gmp (this is a built-in extension as of PHP 7.2+)

```
include "ccxt.php";
var_dump (\ccxt\Exchange::$exchanges); // print a list of all available exchange␣
↪classes
```

15.5.1 Documentation

Read the Manual for more details.

15.5.2 Usage

15.6 Intro

The ccxt library consists of a public part and a private part. Anyone can use the public part out-of-the-box immediately after installation. Public APIs open access to public information from all exchange markets without registering user accounts and without having API keys.

Public APIs include the following:

- market data
- instruments/trading pairs
- price feeds (exchange rates)
- order books
- trade history
- tickers
- OHLC(V) for charting
- other public endpoints

For trading with private APIs you need to obtain API keys from/to exchange markets. It often means registering with exchanges and creating API keys with your account. Most exchanges require personal info or identification. Some kind of verification may be necessary as well. If you want to trade you need to register yourself, this library will not create accounts or API keys for you. Some exchange APIs expose interface methods for registering an account from within the code itself, but most of exchanges don't. You have to sign up and create API keys with their websites.

Private APIs allow the following:

- manage personal account info

- query account balances
- trade by making market and limit orders
- deposit and withdraw fiat and crypto funds
- query personal orders
- get ledger history
- transfer funds between accounts
- use merchant services

This library implements full public and private REST APIs for all exchanges. WebSocket and FIX implementations in JavaScript, PHP, Python and other languages coming soon.

The ccxt library supports both camelcase notation (preferred in JavaScript) and underscore notation (preferred in Python and PHP), therefore all methods can be called in either notation or coding style in any language.

```
// both of these notations work in JavaScript/Python/PHP
exchange.methodName ()  // camelcase pseudocode
exchange.method_name () // underscore pseudocode
```

Read the Manual for more details.

15.7 JavaScript

```
'use strict';
const ccxt = require ('ccxt');

(async function () {
    let kraken    = new ccxt.kraken ()
    let bitfinex  = new ccxt.bitfinex ({ verbose: true })
    let huobi     = new ccxt.huobi ()
    let okcoinusd = new ccxt.okcoinusd ({
        apiKey: 'YOUR_PUBLIC_API_KEY',
        secret: 'YOUR_SECRET_PRIVATE_KEY',
    })

    const exchangeId = 'binance'
        , exchangeClass = ccxt[exchangeId]
        , exchange = new exchangeClass ({
            'apiKey': 'YOUR_API_KEY',
            'secret': 'YOUR_SECRET',
            'timeout': 30000,
            'enableRateLimit': true,
        })

    console.log (kraken.id,    await kraken.loadMarkets ())
    console.log (bitfinex.id,  await bitfinex.loadMarkets  ())
    console.log (huobi.id,     await huobi.loadMarkets ())

    console.log (kraken.id,    await kraken.fetchOrderBook (kraken.symbols[0]))
    console.log (bitfinex.id,  await bitfinex.fetchTicker ('BTC/USD'))
    console.log (huobi.id,     await huobi.fetchTrades ('ETH/CNY'))

    console.log (okcoinusd.id, await okcoinusd.fetchBalance ())
```

(continues on next page)

(continued from previous page)

```
    // sell 1 BTC/USD for market price, sell a bitcoin for dollars immediately
    console.log (okcoinusd.id, await okcoinusd.createMarketSellOrder ('BTC/USD', 1))

    // buy 1 BTC/USD for $2500, you pay $2500 and receive ฿1 when the order is closed
    console.log (okcoinusd.id, await okcoinusd.createLimitBuyOrder ('BTC/USD', 1,␣
↪2500.00))

    // pass/redefine custom exchange-specific order params: type, amount, price or␣
↪whatever
    // use a custom order type
    bitfinex.createLimitSellOrder ('BTC/USD', 1, 10, { 'type': 'trailing-stop' })

}) ();
```

15.8 Python

```
# coding=utf-8

import ccxt

hitbtc = ccxt.hitbtc({'verbose': True})
bitmex = ccxt.bitmex()
huobi  = ccxt.huobi()
exmo   = ccxt.exmo({
    'apiKey': 'YOUR_PUBLIC_API_KEY',
    'secret': 'YOUR_SECRET_PRIVATE_KEY',
})
kraken = ccxt.kraken({
    'apiKey': 'YOUR_PUBLIC_API_KEY',
    'secret': 'YOUR_SECRET_PRIVATE_KEY',
})

exchange_id = 'binance'
exchange_class = getattr(ccxt, exchange_id)
exchange = exchange_class({
    'apiKey': 'YOUR_API_KEY',
    'secret': 'YOUR_SECRET',
    'timeout': 30000,
    'enableRateLimit': True,
})

hitbtc_markets = hitbtc.load_markets()

print(hitbtc.id, hitbtc_markets)
print(bitmex.id, bitmex.load_markets())
print(huobi.id, huobi.load_markets())

print(hitbtc.fetch_order_book(hitbtc.symbols[0]))
print(bitmex.fetch_ticker('BTC/USD'))
print(huobi.fetch_trades('LTC/CNY'))

print(exmo.fetch_balance())
```

(continues on next page)

(continued from previous page)

```
# sell one Ƀ for market price and receive $ right now
print(exmo.id, exmo.create_market_sell_order('BTC/USD', 1))

# limit buy BTC/EUR, you pay €2500 and receive Ƀ1  when the order is closed
print(exmo.id, exmo.create_limit_buy_order('BTC/EUR', 1, 2500.00))

# pass/redefine custom exchange-specific order params: type, amount, price, flags,␣
↪etc...
kraken.create_market_buy_order('BTC/USD', 1, {'trading_agreement': 'agree'})
```

15.9 PHP

```
include 'ccxt.php';

$poloniex = new \ccxt\poloniex ();
$bittrex  = new \ccxt\bittrex  (array ('verbose' => true));
$quoinex  = new \ccxt\quoinex   ();
$zaif     = new \ccxt\zaif     (array (
    'apiKey' => 'YOUR_PUBLIC_API_KEY',
    'secret' => 'YOUR_SECRET_PRIVATE_KEY',
));
$hitbtc   = new \ccxt\hitbtc   (array (
    'apiKey' => 'YOUR_PUBLIC_API_KEY',
    'secret' => 'YOUR_SECRET_PRIVATE_KEY',
));

$exchange_id = 'binance';
$exchange_class = "\\ccxt\\$exchange_id";
$exchange = new $exchange_class (array (
    'apiKey' => 'YOUR_API_KEY',
    'secret' => 'YOUR_SECRET',
    'timeout' => 30000,
    'enableRateLimit' => true,
));

$poloniex_markets = $poloniex->load_markets ();

var_dump ($poloniex_markets);
var_dump ($bittrex->load_markets ());
var_dump ($quoinex->load_markets ());

var_dump ($poloniex->fetch_order_book ($poloniex->symbols[0]));
var_dump ($bittrex->fetch_trades ('BTC/USD'));
var_dump ($quoinex->fetch_ticker ('ETH/EUR'));
var_dump ($zaif->fetch_ticker ('BTC/JPY'));

var_dump ($zaif->fetch_balance ());

// sell 1 BTC/JPY for market price, you pay ¥ and receive Ƀ immediately
var_dump ($zaif->id, $zaif->create_market_sell_order ('BTC/JPY', 1));

// buy BTC/JPY, you receive Ƀ1 for ¥285000 when the order closes
var_dump ($zaif->id, $zaif->create_limit_buy_order ('BTC/JPY', 1, 285000));
```

(continues on next page)

(continued from previous page)

```
// set a custom user-defined id to your order
$hitbtc->create_order ('BTC/USD', 'limit', 'buy', 1, 3000, array ('clientOrderId' =>
↪'123'));
```

15.9.1 Contributing

Please read the CONTRIBUTING document before making changes that you would like adopted in the code. Also, read the Manual for more details.

15.9.2 Support Developer Team

We are investing a significant amount of time into the development of this library. If CCXT made your life easier and you like it and want to help us improve it further or if you want to speed up new features and exchanges, please, support us with a tip. We appreciate all contributions!

15.10 Sponsors

Support this project by becoming a sponsor. Your logo will show up here with a link to your website.

[Become a sponsor]

15.11 Backers

Thank you to all our backers! [Become a backer]

15.12 Crypto

```
ETH 0xa7c2b18b7c8b86984560cad3b1bc3224b388ded0
BTC 33RmVRfhK2WZVQR1R83h2e9yXoqRNDvJva
BCH 1GN9p233TvNcNQFthCgfiHUnj5JRKEc2Ze
LTC LbT8mkAqQBphc4yxLXEDgYDfEax74et3bP
```

Thank you!

15.12.1 Social

- Follow us on Twitter
- Read our blog on Medium

The **CCXT** library is used to connect and trade with cryptocurrency / altcoin exchanges and payment processing services worldwide. It provides quick access to market data for storage, analysis, visualization, indicator development, algorithmic trading, strategy backtesting, bot programming, webshop integration and related software engineering.

It is intended to be used by **coders, developers, technically-skilled traders, data-scientists and financial analysts** for building trading algorithms on top of it.

Current featurelist:

- support for many exchange markets, even more upcoming soon
- fully implemented public and private APIs for all exchanges
- all currencies, altcoins and symbols, prices, order books, trades, tickers, etc. . .
- optional normalized data for cross-exchange or cross-currency analytics and arbitrage
- an out-of-the box unified all-in-one API extremely easy to integrate
- works in Node 7.6+, Python 2 and 3, PHP 5.3+, web browsers

ccxt on GitHub | *Install* | *Usage* | Manual | Examples | Changelog | Contributing

CHAPTER 16

Supported Cryptocurrency Exchange Markets

The ccxt library currently supports the following 91 cryptocurrency exchange markets and trading APIs:

	id	name	ver	doc	countries
	_1broker	1Broker	2	API	US
	_1btcxe	1BTCXE	*	API	Panama
	acx	ACX	2	API	Australia
	allcoin	Allcoin	1	API	Canada
	anxpro	ANXPro	2	API	Japan, Singapore, Hong Kong, New Zealand
	binance	Binance	1	API	China
	bit2c	Bit2C	*	API	Israel
	bitbay	BitBay	*	API	Poland, EU
	bitcoincoid	Bitcoin.co.id	*	API	Indonesia
	bitfinex	Bitfinex	1	API	US
	bitfinex2	Bitfinex v2	2	API	US

Continued on next page

Table 1 – continued from previous page

	id	name	ver	doc	countries
	bitflyer	bitFlyer	1	API	Japan
	bithumb	Bithumb	*	API	South Korea
	bitlish	bitlish	1	API	UK, EU, Russia
	bitmarket	BitMarket	*	API	Poland, EU
	bitmex	BitMEX	1	API	Seychelles
	bitso	Bitso	3	API	Mexico
	bitstamp1	Bitstamp v1	1	API	UK
	bitstamp	Bitstamp	2	API	UK
	bittrex	Bittrex	1.1	API	US
	bl3p	BL3P	1	API	Netherlands, EU
	bleutrade	Bleutrade	2	API	Brazil
	btcbox	BtcBox	1	API	Japan
	btcchina	BTCChina	1	API	China
	btcexchange	BTCExchange	*	API	Philippines
	btcmarkets	BTC Markets	*	API	Australia
	btctradeua	BTC Trade UA	*	API	Ukraine
	btcturk	BTCTurk	*	API	Turkey
	btcx	BTCX	1	API	Iceland, US, EU
	bter	Bter	2	API	British Virgin Islands, China
	bxinth	BX.in.th	*	API	Thailand
	ccex	C-CEX	*	API	Germany, EU

Continued on next page

Table 1 – continued from previous page

	id	name	ver	doc	countries
	cex	CEX.IO	*	API	UK, EU, Cyprus, Russia
	chbtc	CHBTC	1	API	China
	chilebit	ChileBit	1	API	Chile
	coincheck	coincheck	*	API	Japan, Indonesia
	coinfloor	coinfloor	*	API	UK
	coingi	Coingi	*	API	Panama, Bulgaria, China, US
	coinmarketcap	CoinMarketCap	1	API	US
	coinmate	CoinMate	*	API	UK, Czech Republic
	coinsecure	Coinsecure	1	API	India
	coinspot	CoinSpot	*	API	Australia
	cryptopia	Cryptopia	*	API	New Zealand
	dsx	DSX	3	API	UK
	exmo	EXMO	1	API	Spain, Russia
	flowbtc	flowBTC	1	API	Brazil
	foxbit	FoxBit	1	API	Brazil
	fybse	FYB-SE	*	API	Sweden
	fybsg	FYB-SG	*	API	Singapore
	gatecoin	Gatecoin	*	API	Hong Kong
	gateio	Gate.io	2	API	China
	gdax	GDAX	*	API	US
	gemini	Gemini	1	API	US

Continued on next page

Table 1 – continued from previous page

	id	name	ver	doc	countries
	hitbtc	HitBTC	1	API	Hong Kong
	hitbtc2	HitBTC v2	2	API	Hong Kong
	huobi	Huobi	3	API	China
	huobicny	Huobi CNY	1	API	China
	huobipro	Huobi Pro	1	API	China
	independentreserve	Independent Reserve	*	API	Australia, New Zealand
	itbit	itBit	1	API	US
	jubi	jubi.com	1	API	China
	kraken	Kraken	0	API	US
	kuna	Kuna	2	API	Ukraine
	lakebtc	LakeBTC	2	API	US
	livecoin	LiveCoin	*	API	US, UK, Russia
	liqui	Liqui	3	API	Ukraine
	luno	luno	1	API	UK, Singapore, South Africa
	mercado	Mercado Bitcoin	3	API	Brazil
	mixcoins	MixCoins	1	API	UK, Hong Kong
	nova	Novaexchange	2	API	Tanzania
	okcoincny	OKCoin CNY	1	API	China
	okcoinusd	OKCoin USD	1	API	China, US
	okex	OKEX	1	API	China, US
	paymium	Paymium	1	API	France, EU

Continued on next page

Table 1 – continued from previous page

	id	name	ver	doc	countries
	poloniex	Poloniex	*	API	US
	quadrigacx	QuadrigaCX	2	API	Canada
	qryptos	QRYPTOS	2	API	China, Taiwan
	quoine	QUOINE	2	API	Japan, Singapore, Vietnam
	southxchange	SouthXchange	*	API	Argentina
	surbitcoin	SurBitcoin	1	API	Venezuela
	tidex	Tidex	3	API	UK
	therock	TheRockTrading	1	API	Malta
	urdubit	UrduBit	1	API	Pakistan
	vaultoro	Vaultoro	1	API	Switzerland
	vbtc	VBTC	1	API	Vietnam
	virwox	VirWoX	*	API	Austria, EU
	wex	WEX	3	API	New Zealand
	xbtce	xBTCe	1	API	Russia
	yobit	YoBit	3	API	Russia
	yunbi	YUNBI	2	API	China
	zaif	Zaif	1	API	Japan

The list above is updated frequently, new crypto markets, altcoin exchanges, bug fixes, API endpoints are introduced and added on regular basis. See the Manual for details. If you don't find a cryptocurrency exchange market in the list above and/or want another exchange to be added, post or send us a link to it by opening an issue here on GitHub or via email.

The library is under MIT license, that means it's absolutely free for any developer to build commercial and opensource software on top of it, but use it at your own risk with no warranties, as is.

CHAPTER 17

Install

The easiest way to install the ccxt library is to use builtin package managers:

- ccxt in **NPM** (JavaScript / Node v7.6+)
- ccxt in **PyPI** (Python 2 and 3)

This library is shipped as an all-in-one module implementation with minimalistic dependencies and requirements:

- `js/ <https://github.com/ccxt/ccxt/blob/master/js/>`__ in JavaScript
- `python/ <https://github.com/ccxt/ccxt/blob/master/python/>`__ in Python (generated from JS)
- `php/ <https://github.com/ccxt/ccxt/blob/master/php/>`__ in PHP (generated from JS)

You can also clone it into your project directory from ccxt GitHub repository:

```
git clone https://github.com/ccxt/ccxt.git
```

An alternative way of installing this library into your code is to copy a single file manually into your working directory with language extension appropriate for your environment.

17.1 JavaScript (NPM)

JavaScript version of CCXT works both in Node and web browsers. Requires ES6 and `async/await` syntax support (Node 7.6.0+). When compiling with Webpack and Babel, make sure it is not excluded in your `babel-loader` config.

ccxt in **NPM**

```
npm install ccxt
```

```
var ccxt = require ('ccxt')

console.log (ccxt.exchanges) // print all available exchanges
```

17.2 JavaScript (for use with the <script> tag):

All-in-one browser bundle (dependencies included), served from unpkg CDN, which is a fast, global content delivery network for everything on NPM.

```
<script type="text/javascript" src="https://unpkg.com/ccxt"></script>
```

Creates a global `ccxt` object:

```
console.log (ccxt.exchanges) // print all available exchanges
```

17.3 Python

ccxt in **PyPI**

```
pip install ccxt
```

```
import ccxt
print(ccxt.exchanges) # print a list of all available exchange classes
```

The library supports concurrent asynchronous mode with asyncio and async/await in Python 3.5+

```
import ccxt.async as ccxt # link against the asynchronous version of ccxt
```

17.4 PHP

The ccxt library in PHP: **‘‘ccxt.php‘‘**

It requires common PHP modules:

- cURL
- mbstring (using UTF-8 is highly recommended)
- PCRE
- iconv

```
include "ccxt.php";
var_dump (\ccxt\Exchange::$exchanges); // print a list of all available exchange␣
↪classes
```

CHAPTER 18

Documentation

Read the Manual for more details.

CHAPTER 19

Usage

19.1 Intro

The ccxt library consists of a public part and a private part. Anyone can use the public part out-of-the-box immediately after installation. Public APIs open access to public information from all exchange markets without registering user accounts and without having API keys.

Public APIs include the following:

- market data
- instruments/trading pairs
- price feeds (exchange rates)
- order books
- trade history
- tickers
- OHLC(V) for charting
- other public endpoints

For trading with private APIs you need to obtain API keys from/to exchange markets. It often means registering with exchanges and creating API keys with your account. Most exchanges require personal info or identification. Some kind of verification may be necessary as well. If you want to trade you need to register yourself, this library will not create accounts or API keys for you. Some exchange APIs expose interface methods for registering an account from within the code itself, but most of exchanges don't. You have to sign up and create API keys with their websites.

Private APIs allow the following:

- manage personal account info
- query account balances
- trade by making market and limit orders
- deposit and withdraw fiat and crypto funds

- query personal orders
- get ledger history
- transfer funds between accounts
- use merchant services

This library implements full public and private REST APIs for all exchanges. WebSocket and FIX implementations in JavaScript, PHP, Python and other languages coming soon.

The ccxt library supports both camelcase notation (preferred in JavaScript) and underscore notation (preferred in Python and PHP), therefore all methods can be called in either notation or coding style in any language.

```
// both of these notations work in JavaScript/Python/PHP
exchange.methodName ()  // camelcase pseudocode
exchange.method_name () // underscore pseudocode
```

Read the Manual for more details.

19.2 JavaScript

```
'use strict';
var ccxt = require ('ccxt')

;(() => async function () {

    let kraken    = new ccxt.kraken ()
    let bitfinex  = new ccxt.bitfinex ({ verbose: true })
    let huobi     = new ccxt.huobi ()
    let okcoinusd = new ccxt.okcoinusd ({
        apiKey: 'YOUR_PUBLIC_API_KEY',
        secret: 'YOUR_SECRET_PRIVATE_KEY',
    })

    let krakenMarkets = await kraken.loadMarkets ()

    console.log (kraken.id,    krakenMarkets)
    console.log (bitfinex.id,  await bitfinex.loadMarkets  ())
    console.log (huobi.id,     await huobi.loadMarkets ())

    console.log (kraken.id,    await kraken.fetchOrderBook (kraken.symbols[0]))
    console.log (bitfinex.id,  await bitfinex.fetchTicker ('BTC/USD'))
    console.log (huobi.id,     await huobi.fetchTrades ('ETH/CNY'))

    console.log (okcoinusd.id, await okcoinusd.fetchBalance ())

    // sell 1 BTC/USD for market price, sell a bitcoin for dollars immediately
    console.log (okcoinusd.id, await okcoinusd.createMarketSellOrder ('BTC/USD', 1))

    // buy 1 BTC/USD for $2500, you pay $2500 and receive ฿1 when the order is closed
    console.log (okcoinusd.id, await okcoinusd.createLimitBuyOrder ('BTC/USD', 1,␣
↪2500.00))

    // pass/redefine custom exchange-specific order params: type, amount, price or␣
↪whatever
    // use a custom order type
```

(continues on next page)

(continued from previous page)

```
    bitfinex.createLimitSellOrder ('BTC/USD', 1, 10, { 'type': 'trailing-stop' })
}) ()
```

19.3 Python

```
# coding=utf-8

import ccxt

hitbtc = ccxt.hitbtc({'verbose': True})
bitmex = ccxt.bitmex()
huobi  = ccxt.huobi()
exmo   = ccxt.exmo({
    'apiKey': 'YOUR_PUBLIC_API_KEY',
    'secret': 'YOUR_SECRET_PRIVATE_KEY',
})

hitbtc_markets = hitbtc.load_markets()

print(hitbtc.id, hitbtc_markets)
print(bitmex.id, bitmex.load_markets())
print(huobi.id, huobi.load_markets())

print(hitbtc.fetch_order_book(hitbtc.symbols[0]))
print(bitmex.fetch_ticker('BTC/USD'))
print(huobi.fetch_trades('LTC/CNY'))

print(exmo.fetch_balance())

# sell one Ƀ for market price and receive $ right now
print(exmo.id, exmo.create_market_sell_order('BTC/USD', 1))

# limit buy BTC/EUR, you pay €2500 and receive Ƀ1  when the order is closed
print(exmo.id, exmo.create_limit_buy_order('BTC/EUR', 1, 2500.00))

# pass/redefine custom exchange-specific order params: type, amount, price, flags,␣
↪etc...
kraken.create_market_buy_order('BTC/USD', 1, {'trading_agreement': 'agree'})
```

19.4 PHP

```
include 'ccxt.php';

$poloniex = new \ccxt\poloniex  ();
$bittrex  = new \ccxt\bittrex   (array ('verbose' => true));
$quoine   = new \ccxt\zaif      ();
$zaif     = new \ccxt\quoine    (array (
    'apiKey' => 'YOUR_PUBLIC_API_KEY',
    'secret' => 'YOUR_SECRET_PRIVATE_KEY',
));
```

(continues on next page)

(continued from previous page)

```
$poloniex_markets = $poloniex->load_markets ();

var_dump ($poloniex_markets);
var_dump ($bittrex->load_markets ());
var_dump ($quoine->load_markets ());

var_dump ($poloniex->fetch_order_book ($poloniex->symbols[0]));
var_dump ($bittrex->fetch_trades ('BTC/USD'));
var_dump ($quoine->fetch_ticker ('ETH/EUR'));
var_dump ($zaif->fetch_ticker ('BTC/JPY'));

var_dump ($zaif->fetch_balance ());

// sell 1 BTC/JPY for market price, you pay ¥ and receive ₿ immediately
var_dump ($zaif->id, $zaif->create_market_sell_order ('BTC/JPY', 1));

// buy BTC/JPY, you receive ₿1 for ¥285000 when the order closes
var_dump ($zaif->id, $zaif->create_limit_buy_order ('BTC/JPY', 1, 285000));

// set a custom user-defined id to your order
$hitbtc->create_order ('BTC/USD', 'limit', 'buy', 1, 3000, array ('clientOrderId' =>
↪'123'));
```

CHAPTER 20

Contributing

Please read the CONTRIBUTING document before making changes that you would like adopted in the code. Also, read the Manual for more details.