ccxt Documentation

Release 1.10.730

Igor Kroitor

Jan 16, 2018

General Information

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 98 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
	bibox	Bibox	1	API	China, US, South Korea
	binance	Binance	*	API	Japan
	bit2c	Bit2C	*	API	Israel
	bitbay	BitBay	*	API	Poland, EU
	bitcoincoid	Bitcoin.co.id	1.7	API	Indonesia
	bitfinex	Bitfinex	1	API	British Virgin Islands

Continued on next page

Table 1.1 – continued from previous page

	id	name	ver	doc	countries
	bitfinex2	Bitfinex v2	2	API	British Virgin Islands
	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
	bitstamp	Bitstamp	2	API	UK
	bitstamp1	Bitstamp v1	1	API	UK
	bittrex	Bittrex	1.1	API	US
	bl3p	BL3P	1	API	Netherlands, EU
	bleutrade	Bleutrade	2	API	Brazil
	braziliex	Braziliex	*	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

Continued on next page

Table 1.1 – continued from previous page

	id	name	ver	doc	countries
	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
	coincheck	coincheck	*	API	Japan, Indonesia
	coinexchange	CoinExchange	*	API	India, Japan, South Korea, Vietnam, US
	coinfloor	coinfloor	*	API	UK
	coingi	Coingi	*	API	Panama, Bulgaria, China, US
	coinmarketcap	CoinMarketCap	1	API	US
	coinmate	CoinMate	*	API	UK, Czech Republic, EU
	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

Continued on next page

Table 1.1 – continued from previous page

	id	name	ver	doc	countries
	gateio	Gate.io	2	API	China
	gdax	GDAX	*	API	US
	gemini	Gemini	1	API	US
	getbtc	GetBTC	*	API	St. Vincent & Grenadines, Russia
	hitbtc	HitBTC	1	API	UK
	hitbtc2	HitBTC v2	2	API	UK
	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
	kucoin	Kucoin	1	API	Hong Kong
	kuna	Kuna	2	API	Ukraine
	lakebtc	LakeBTC	2	API	US
	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

Continued on next page

Table 1.1 – continued from previous page

	id	name	ver	doc	countries
	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
	poloniex	Poloniex	*	API	US
	qryptos	QRYPTOS	2	API	China, Taiwan
	quadrigacx	QuadrigaCX	2	API	Canada
	quoine	QUOINE	2	API	Japan, Singapore, Vietnam
	southxchange	SouthXchange	*	API	Argentina
	surbitcoin	SurBitcoin	1	API	Venezuela
	therock	TheRockTrading	1	API	Malta
	tidex	Tidex	3	API	UK
	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

Continued on next page

Table 1.1 – continued from previous page

	id	name	ver	doc	countries
	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	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 2.1 – continued from previous page

country / region	logo	id	name	ver	doc
British Virgin Islands	BITFINEX	bitfinex	Bitfinex	1	API
British Virgin Islands	BITFINEX	bitfinex2	Bitfinex v2	2	API
British Virgin Islands	Bter.com	bter	Bter	2	API
Bulgaria	coingi.com	coingi	Coingi	*	API
Canada	ALLCOIN	allcoin	Allcoin	1	API
Canada	QUADRIGACX	quadrigacx	QuadrigaCX	2	API
Chile	CHILEBIT.NET	chilebit	ChileBit	1	API
China	Bibox	bibox	Bibox	1	API
China	比特币中国 BTCCHINA.COM	btcchina	BTCChina	1	API
China	Bter.com	bter	Bter	2	API
China	CHBTC	chbtc	CHBTC	1	API
China	coingi.com	coingi	Coingi	*	API
China	gate.io	gateio	Gate.io	2	API
China	火币 huobi.com	huobi	Huobi	3	API
China	火币 huobi.com	huobicny	Huobi CNY	1	API
China	火币 huobi.com	huobipro	Huobi Pro	1	API
China	聚币网 jubi.com	jubi	jubi.com	1	API
China	OKCoin CNY	okcoincny	OKCoin CNY	1	API
China	OKCoin USD	okcoinusd	OKCoin USD	1	API
China	OKEX	okex	OKEX	1	API
China	QRYPTOS	qryptos	QRYPTOS	2	API

Continued on next page

Table 2.1 – continued from previous page

country / region	logo	id	name	ver	doc
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
EU		cex	CEX.IO	*	API
EU		coinmate	CoinMate	*	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		gatecoin	Gatecoin	*	API
Hong Kong		kucoin	Kucoin	1	API
Hong Kong		mixcoins	MixCoins	1	API
Iceland		btcx	BTCX	1	API

Continued on next page

Table 2.1 – continued from previous page

country / region	logo	id	name	ver	doc
India		coinexchange	CoinExchange	*	API
India		coinsecure	Coinsecure	1	API
Indonesia		bitcoincoid	Bitcoin.co.id	1.7	API
Indonesia		coincheck	coincheck	*	API
Israel		bit2c	Bit2C	*	API
Japan		anxpro	ANXPro	2	API
Japan		binance	Binance	*	API
Japan		bitflyer	bitFlyer	1	API
Japan		btcbox	BtcBox	1	API
Japan		coincheck	coincheck	*	API
Japan		coinexchange	CoinExchange	*	API
Japan		quoine	QUOINE	2	API
Japan		zaif	Zaif	1	API
Malta		therock	TheRockTrading	1	API
Mexico		bitso	Bitso	3	API
Netherlands		bl3p	BL3P	1	API
New Zealand		anxpro	ANXPro	2	API
New Zealand		cryptopia	Cryptopia	*	API
New Zealand		independentreserve	Independent Reserve	*	API
New Zealand		wex	WEX	3	API
Pakistan		urdubit	UrduBit	1	API

Continued on next page

Table 2.1 – continued from previous page

country / region	logo	id	name	ver	doc
Panama		_1btcxe	1BTCXE	*	API
Panama		coingi	Coingi	*	API
Philippines		btcexchange	BTCExchange	*	API
Poland		bitbay	BitBay	*	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		quoine	QUOINE	2	API
South Africa		luno	luno	1	API
South Korea		bibox	Bibox	1	API
South Korea		bithumb	Bithumb	*	API
South Korea		coinexchange	CoinExchange	*	API

Continued on next page

Table 2.1 – continued from previous page

country / region	logo	id	name	ver	doc
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		qryptos	QRYPTOS	2	API
Tanzania		nova	Novaexchange	2	API
Thailand		bxinth	BX.in.th	*	API
Turkey		btcturk	BTCTurk	*	API
UK		bitlish	Bitlish	1	API
UK		bitstamp	Bitstamp	2	API
UK		bitstamp1	Bitstamp v1	1	API
UK		cex	CEX.IO	*	API
UK		coinfloor	coinfloor	*	API
UK		coinmate	CoinMate	*	API
UK		dsx	DSX	3	API
UK		hitbtc	HitBTC	1	API
UK		hitbtc2	HitBTC v2	2	API
UK		livecoin	LiveCoin	*	API
UK		luno	luno	1	API
UK		mixcoins	MixCoins	1	API

Continued on next page

Table 2.1 – continued from previous page

country / region	logo	id	name	ver	doc
UK		tidex	Tidex	3	API
Ukraine		btctradeua	BTC Trade UA	*	API
Ukraine		kuna	Kuna	2	API
Ukraine		liqui	Liqui	3	API
US		_1broker	1Broker	2	API
US		bibox	Bibox	1	API
US		bittrex	Bittrex	1.1	API
US		btcx	BTCX	1	API
US		coinexchange	CoinExchange	*	API
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	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
Venezuela		surbitcoin	SurBitcoin	1	API

Continued on next page

Table 2.1 – continued from previous page

country / region	logo	id	name	ver	doc
Vietnam		coinexchange	CoinExchange	*	API
Vietnam		quoine	QUOINE	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/kroitor/ccxt/blob/master/ccxt.js>‘__ in JavaScript
- `./python/ <https://github.com/kroitor/ccxt/blob/master/python/>‘__ in Python (generated from JS)
- `ccxt.php <https://github.com/kroitor/ccxt/blob/master/ccxt.php>‘__ in PHP (generated from JS)

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

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

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
```

3.2 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+

```
import ccxt.async 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``**

It requires common PHP modules:

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

```
include "ccxt.php";
var_dump (\cxxt\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.

If you need a 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.

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

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/'`

4.1 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',
        'https': 'http://10.10.1.10:1080',
    },
})
```

Or

```
import ccxt
exchange = ccxt.poloniex()
exchange.proxies = {
  'http': 'http://10.10.1.10:3128',
  'https': 'http://10.10.1.10:1080',
}
```

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)

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, like https://crossorigin.me.

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       |
|       fetchTicker            .            cancelOrder       |
|       fetchTickers           .             fetchOrder       |
|       fetchOrderBook         .            fetchOrders       |
|       fetchOHLCV             .        fetchOpenOrders       |
|       fetchTrades            .      fetchClosedOrders       |
|                              .          fetchMyTrades       |
|                              .                deposit       |
|                              .               withdraw       |
│                              .                              |
+=============================================================+
│                              .                              |
|                     Custom Exchange API                     |
|                      (Derived Classes)                      |
│                              .                              |
|       publicGet...           .          privateGet...       |
|       publicPost...          .         privatePost...       |
|                              .          privatePut...       |
|                               .       privateDelete...       |
|                               .                   sign       |
|                               .                              |
+=============================================================+
|                               .                              |
|                       Base Exchange Class                    |
|                               .                              |
+=============================================================+
```

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 98 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
	bibox	Bibox	1	API	China, US, South Korea
	binance	Binance	*	API	Japan
	bit2c	Bit2C	*	API	Israel
	bitbay	BitBay	*	API	Poland, EU
	bitcoincoid	Bitcoin.co.id	1.7	API	Indonesia
	bitfinex	Bitfinex	1	API	British Virgin Islands

Continued on next page

Table 7.1 – continued from previous page

	id	name	ver	doc	countries
	bitfinex2	Bitfinex v2	2	API	British Virgin Islands
	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
	bitstamp	Bitstamp	2	API	UK
	bitstamp1	Bitstamp v1	1	API	UK
	bittrex	Bittrex	1.1	API	US
	bl3p	BL3P	1	API	Netherlands, EU
	bleutrade	Bleutrade	2	API	Brazil
	braziliex	Braziliex	*	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

Continued on next page

Table 7.1 – continued from previous page

	id	name	ver	doc	countries
	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
	coincheck	coincheck	*	API	Japan, Indonesia
	coinexchange	CoinExchange	*	API	India, Japan, South Korea, Vietnam, US
	coinfloor	coinfloor	*	API	UK
	coingi	Coingi	*	API	Panama, Bulgaria, China, US
	coinmarketcap	CoinMarketCap	1	API	US
	coinmate	CoinMate	*	API	UK, Czech Republic, EU
	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

Continued on next page

Table 7.1 – continued from previous page

	id	name	ver	doc	countries
	gateio	Gate.io	2	API	China
	gdax	GDAX	*	API	US
	gemini	Gemini	1	API	US
	getbtc	GetBTC	*	API	St. Vincent & Grenadines, Russia
	hitbtc	HitBTC	1	API	UK
	hitbtc2	HitBTC v2	2	API	UK
	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
	kucoin	Kucoin	1	API	Hong Kong
	kuna	Kuna	2	API	Ukraine
	lakebtc	LakeBTC	2	API	US
	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

Continued on next page

Table 7.1 – continued from previous page

	id	name	ver	doc	countries
	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
	poloniex	Poloniex	*	API	US
	qryptos	QRYPTOS	2	API	China, Taiwan
	quadrigacx	QuadrigaCX	2	API	Canada
	quoine	QUOINE	2	API	Japan, Singapore, Vietnam
	southxchange	SouthXchange	*	API	Argentina
	surbitcoin	SurBitcoin	1	API	Venezuela
	therock	TheRockTrading	1	API	Malta
	tidex	Tidex	3	API	UK
	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

Continued on next page

Table 7.1 – continued from previous page

	id	name	ver	doc	countries
	yunbi	YUNBI	2	API	China
	zaif	Zaif	1	API	Japan
	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
import ccxt
print (ccxt.exchanges)
```

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

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

```
// 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] ();
```

```
# 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') ()
```

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
date_default_timezone_set ('UTC');
include 'ccxt.php';
$bitfinex = new \ccxt\bitfinex (); // default id
$bitfinex1 = new \ccxt\bitfinex (array ('id' => 'bitfinex1'));
$bitfinex2 = new \ccxt\bitfinex (array ('id' => 'bitfinex2'));
$id = 'kraken';
$kraken = new \ccxt\$id ();
```

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' ],   // string or 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
    'hasFetchTickers':  true,            // true if the exchange implements␣
↪fetchTickers ()
    'hasFetchOHLCV':    false,           // true if the exchange implements fetchOHLCV␣
↪()
    'timeframes': {                      // empty if the exchange !hasFetchOHLCV
        '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`: A string literal or 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.
- `hasFetchTickers`: This is a boolean property indicating if the exchange has the fetchTickers () method available. When this property is false, the exchange will also throw a NotSupported exception upon a call to fetchTickers ().
- `hasFetchOHLCV`: This is a boolean property indicating if the exchange has the fetchOHLCV () method available. When this property is false, the exchange will also throw a NotSupported exception upon a call to fetchOHLCV (). Also, if this property is true, the `timeframes` property is populated as well.
- `timeframes`: An associative array of timeframes, supported by the fetchOHLCV method of the exchange. This is only populated when `hasFetchTickers` 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. This parameter is not used for now (reserved for future).
- `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).
- `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.

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.

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.

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

- try later
- 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
- ...

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

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).

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';
$huobi = new \ccxt\$id ();
$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 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`.

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 parameter. 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.marketsById['XRPBTC']                // id → market (get market by id)

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

})
```

```
# 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

$var_dump ($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`.

8.3.2 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**.

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
# 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
// 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 cancelling 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

In the code for each exchange, you'll notice that functions that make API requests aren't explicitly defined. This is because the `api` definition in the exchange description JSON is used to create *magic functions* (aka *partial functions* or *closures*) inside the exchange subclass. That implicit injection is done by the `defineRestApi/ define_rest_api` base exchange method.

Each partial function takes a dictionary of `params` and returns the API response. For example, if an exchange offers a HTTP GET URL for querying prices like `https://example.com/public/quotes`, it is converted to a method named `example.publicGetQuotes (params = {}) / $example->publicGetQuotes ($params = array ())`.

Upon instantiation the base exchange class takes each URL from its list of endpoints, splits it into words, and then makes up a callable function name from those words by using a partial construct.

The endpoint 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.

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` subpackage, like in the following example:

```
# Python

import asyncio
import ccxt.async 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[, params])`: Fetch L2/L3 order book for a particular market trading symbol.
- `fetchL2OrderBook (symbol[, 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[, params]])`
- `fetchOpenOrders ([symbol[, params]])`
- `fetchClosedOrders ([symbol[, 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. Use the `params` dictionary if you need to pass a custom setting or an optional parameter to your unified query.

CHAPTER 10

Market Data

10.1 Order Book / Market Depth

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 ],
        [ 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
}
```

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.

Some exchanges accept a second dictionary of extra parameters to the `fetchOrderBook () / fetch_order_book ()` function allowing you to get the level of aggregation you need, like so:

```
// JavaScript

(async function test () {
    const ccxt = require ('ccxt')
    const exchange = new ccxt.bitfinex ()
    const orders = await exchange.fetchOrderBook ('BTC/USD', {
        'limit_bids': 5, // max = 50
        'limit_asks': 5, // may be 0 in which case the array is empty
        '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
ccxt.cex().fetch_order_book('BTC/USD', {'depth': 10})
```

```
// PHP

// instantiate the exchange by id
$exchange = '\\ccxt\\kraken';
$exchange = new $exchange ();
var_dump ($exchange->fetch_order_book ('BTC/USD', array (
    'count' => 10, // up to ten orders on each side for example
)));
```

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, params)` or `fetch_l2_order_book(symbol, params)` unified method for that.

10.1.1 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 bid (buy) price)
    'ask':         float (current ask (sell) price)
    'vwap':        float (volume weighed average price)
    'open':        float (open price),
    'first':       float (price of first trade),
    'last':        float (price of last trade),
    'change':      float (percentage change),
    'average':     float (average),
    'baseVolume':  float (volume of base currency),
    'quoteVolume': float (volume of quote currency),
}
```

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

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
(async () => {
    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
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!)
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 markets (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
(async () => {
    console.log (await (exchange.fetchTickers ())) // all tickers indexed by their␣
↪symbols
}) ()
```

```
# Python
print(exchange.fetch_tickers()) # all tickers indexed by their symbols
```

```
// PHP
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 `hasFetchOHLCV` 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 most recent OHLCV candles for a particular symbol like so:

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

```
# Python
import time
if exchange.hasFetchOHLCV:
    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 ($exchange->hasFetchOHLCV)
    foreach ($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 `hasFetchTickers` 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.

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

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

10.4 Trades, Orders, 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
let sleep = (ms) => new Promise (resolve => setTimeout (resolve, ms));
(async () => {
    for (symbol in exchange.markets) {
        await sleep (exchange.rateLimit) // milliseconds
        console.log (await exchange.fetchTrades (symbol))
    }
}) ()
```

```
# Python
import time
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
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, most recent trade first) 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.

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.

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 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',
})
```

```
# Python

import ccxt

# 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',
})
```

```
// PHP

include 'ccxt.php'
```

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

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

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.

Also, some exchanges cannot return certain fields and are only capable of telling a total balance (without details). Therefore some or all of the free, used and total amounts may be undefined, None or null. You need to account for that when working with returned balances.

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

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

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

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 their ids or statuses, 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 if possible.

By Order Id

To get 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:

```
//  you can use the params argument for custom overrides
exchange.fetchOrder (id, symbol = undefined, params = {})
```

You can pass custom overrided key-values in additional params 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)
order = exchange.fetch_order(id)
print(order)
```

```
# Python 3.5+ asyncio (asynchronous)
import asyncio
import ccxt.async as ccxt
order = asyncio.get_event_loop().run_until_complete(exchange.fetch_order(id))
print(order)
```

```
// PHP
$order = $exchange->fetch_order ($id);
var_dump ($order);
```

All Orders

```
exchange.fetchOrders (symbol = undefined, since = undefined, limit = undefined,␣
↪params = {})
```

Open Orders

```
exchange.fetchOpenOrders (symbol = undefined, since = undefined, limit = undefined,␣
↪params = {})
```

Closed Orders

```
exchange.fetchClosedOrders (symbol = undefined, since = undefined, limit = undefined,␣
↪params = {})
```

Trades / Transactions / Fills / Executions

```
- 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
```

Recent Trades

```
exchange.fetchMyTrades (symbol = undefined, since = undefined, limit = undefined,␣
↪params = {})
```

Trades By Order Id

```
UNDER CONSTRUCTION
```

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 with milliseconds
    'timestamp':  1502962946216, // Unix timestamp in milliseconds
    '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':     1.0,           // filled amount of base currency
    'remaining':  0.5,           // remaining amount to fill
    '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
    },
    'info':      { ... },        // the original unparsed order structure as is
}
```

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...
- `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.

Note, that some exchanges will not accept market orders (they allow limit orders only).

```
// 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])
```

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.

```
// 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 Cancelling 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)
```

11.5 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
```

11.5.1 Deposit

```
UNDER CONSTRUCTION
```

11.5.2 Withdraw

```
exchange.withdraw (currency, amount, address, params = {})
```

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.

11.5.3 Ledger

```
UNDER CONSTRUCTION
```

11.6 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, read ‘API docs for your exchange <https://github.com/ccxt/ccxt/wiki/Exchanges>‘__ 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):
    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
|   |
|   +---+ InsufficientFunds
|   |
|   +---+ InvalidOrder
|       |
|       +---+ OrderNotFound
```

```
|
+---+ NetworkError (recoverable)
    |
    +---+ DDoSProtection
    |
    +---+ TimeoutError
    |
    +---+ ExchangeNotAvailable
```

- `BaseError`: 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.
- `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
  - some additional endpoint parameter required by the exchange is missing
  - the format of some parameters passed into the endpoint is incorrect
  - an exchange replies with an unclear answer
- `NotSupported`: This exception is raised if the endpoint is not offered/not supported by the exchange API.
- `InsufficientFunds`: This exception is raised when you don't have enough currency on your account balance to place an order.
- `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.
- `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`, some times you also need `uid` and/or `password`.
- `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.
  - `DDoSProtection`: This exception is thrown whenever Cloudflare or Incapsula rate limiter restrictions are enforced per user or region/location. The ccxt library does a case-insensitive search in the response received from the exchange for one of the following keywords:
    - `cloudflare`
    - `incapsula`
  - `RequestTimeout`: The name literally says it all. This exception is raised when connection with the exchange fails or data is not fully received in a specified amount of time. This is controlled by the `timeout` option.
  - `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`

CHAPTER 13

Troubleshooting

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

1. Check the CHANGELOG for recent updates.
2. Turn `verbose = true` to get more detail about it.
3. Check you API credentials. Try a fresh new keypair if possible.
4. 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.
5. 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, like show in the long-poller examples, also here.
6. Read the docs for your exchange and compare your verbose output to the docs.
7. Check your connectivity with the exchange by accessing it with your browser.
8. Check your connection with the exchange through a proxy. Read the Proxy section for more details.
9. 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.
10. 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 exchange in details. The verbose output will also be of use for us to debug it if you submit an issue on GitHub.

- 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

CCXT – CryptoCurrency eXchange Trading Library

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

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

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

14.1 Supported Cryptocurrency Exchange Markets

The ccxt library currently supports the following 98 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
	bibox	Bibox	1	API	China, US, South Korea
	binance	Binance	*	API	Japan
	bit2c	Bit2C	*	API	Israel
	bitbay	BitBay	*	API	Poland, EU
	bitcoincoid	Bitcoin.co.id	1.7	API	Indonesia
	bitfinex	Bitfinex	1	API	British Virgin Islands
	bitfinex2	Bitfinex v2	2	API	British Virgin Islands
	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
	bitstamp	Bitstamp	2	API	UK
	bitstamp1	Bitstamp v1	1	API	UK
	bittrex	Bittrex	1.1	API	US

Continued on next page

Table 14.1 – continued from previous page

	id	name	ver	doc	countries
	bl3p	BL3P	1	API	Netherlands, EU
	bleutrade	Bleutrade	2	API	Brazil
	braziliex	Braziliex	*	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
	cex	CEX.IO	*	API	UK, EU, Cyprus, Russia
	chbtc	CHBTC	1	API	China
	chilebit	ChileBit	1	API	Chile
	coincheck	coincheck	*	API	Japan, Indonesia
	coinexchange	CoinExchange	*	API	India, Japan, South Korea, Vietnam, US
	coinfloor	coinfloor	*	API	UK
	coingi	Coingi	*	API	Panama, Bulgaria, China, US
	coinmarketcap	CoinMarketCap	1	API	US

Continued on next page

Table 14.1 – continued from previous page

	id	name	ver	doc	countries
	coinmate	CoinMate	*	API	UK, Czech Republic, EU
	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
	getbtc	GetBTC	*	API	St. Vincent & Grenadines, Russia
	hitbtc	HitBTC	1	API	UK
	hitbtc2	HitBTC v2	2	API	UK
	huobi	Huobi	3	API	China
	huobicny	Huobi CNY	1	API	China
	huobipro	Huobi Pro	1	API	China
	independentreserve	Independent Reserve	*	API	Australia, New Zealand

Continued on next page

Table 14.1 – continued from previous page

	id	name	ver	doc	countries
	itbit	itBit	1	API	US
	jubi	jubi.com	1	API	China
	kraken	Kraken	0	API	US
	kucoin	Kucoin	1	API	Hong Kong
	kuna	Kuna	2	API	Ukraine
	lakebtc	LakeBTC	2	API	US
	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
	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
	quoine	QUOINE	2	API	Japan, Singapore, Vietnam

Continued on next page

Table 14.1 – continued from previous page

	id	name	ver	doc	countries
	southxchange	SouthXchange	*	API	Argentina
	surbitcoin	SurBitcoin	1	API	Venezuela
	therock	TheRockTrading	1	API	Malta
	tidex	Tidex	3	API	UK
	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.

14.2 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)
- 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.

14.2.1 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
```

14.2.2 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
```

14.2.3 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
```

14.2.4 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
```

14.3 Documentation

Read the Manual for more details.

14.4 Usage

14.4.1 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.

14.4.2 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',
    })

    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 ())

    // 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' })
}) ();
```

14.4.3 Python

```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'})
```

14.4.4 PHP

```php
include 'ccxt.php';

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

$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'));
```

14.5 Contributing

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

14.6 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!

14.6.1 Sponsors

Become our sponsor and get your logo on our Github page with a link to your site.

14.6.2 Supporters

Become our supporter with a monthly payment and get your nick/link on our GitHub page.

14.6.3 Backers

Become a backer with a small monthly donation and help us continue our activities.

14.6.4 Crypto

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

Thank you!

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 15

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 15.1 – 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 15.1 – 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 15.1 – 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 15.1 – 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 16

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.

16.1 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
```

16.2 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
```

16.3 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
```

16.4 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 17

Documentation

Read the Manual for more details.

CHAPTER 18

Usage

18.1 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.

18.2 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
    bitfinex.createLimitSellOrder ('BTC/USD', 1, 10, { 'type': 'trailing-stop' })
}) ()
```

18.3 Python

```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'})
```

18.4 PHP

```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',
));

$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 19

Contributing

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