Coinbase Pro Asyncronous Websocket Client Documentation

Release 1.2.9

Tony Podlaski

Jun 30, 2021

Contents

Asyncronous Python REST and WebSocket Clients for Coinbase Pro

---

Quick Links: Documentation - Source Code - PyPi

Related: Coinbase Pro Digital Currency Exchange - Coinbase Pro REST API - Coinbase Pro WebSocket API

CHAPTER 1

Introduction

The CoPrA (**Co**inbase **Pr**o **A**sync) package provides asyncronous REST and WebSocket clients written in Python for use with the Coinbase Pro digital currency trading platform. To learn about Coinbase Pro's REST and WebSocket APIs as well as how to obtain an API key for authentication to those services, please see Coinbase Pro's API documentation.

CHAPTER 2

CoPrA Features

- compatible with Python 3.5 or greater
- utilizes Python's asyncio concurrency framework
- open source (MIT license)

2.1 REST Features

- Asyncronous REST client class with 100% of the account management, trading, and market data functionality offered by the Coinbase Pro REST API.
- supports user authentication
- built on **aiohttp**, the asynchronous HTTP client/server framework for asyncio and Python

2.2 WebSocket Features

- Asyncronous WebSocket client class with callback hooks for managing every phase of a Coinbase Pro WebSocket session
- supports user authentication
- built on **Autobahn|Python**, the open-source (MIT) real-time framework for web, mobile & the Internet of Things.

CHAPTER 3

Examples

3.1 REST

Without a Coinbase Pro API key, `copra.rest.Client` has access to all of the public market data that Coinbase makes available.

```
# 24hour_stats.py

import asyncio

from copra.rest import Client

loop = asyncio.get_event_loop()

client = Client(loop)

async def get_stats():
    btc_stats = await client.get_24hour_stats('BTC-USD')
    print(btc_stats)

loop.run_until_complete(get_stats())
loop.run_until_complete(client.close())
```

Running the above:

```
$ python3 24hour_stats.py
{'open': '3914.96000000', 'high': '3957.10000000', 'low': '3508.00000000', 'volume':
↪'37134.10720409', 'last': '3670.06000000', 'volume_30day': '423047.53794129'}
```

In conjunction with a Coinbase Pro API key, `copra.rest.Client` can be used to trade cryptocurrency and manage your Coinbase pro account. This example also shows how `copra.rest.Client` can be used as a context manager.

```
# btc_account_info.py

```

(continues on next page)

(continued from previous page)

```
import asyncio

from copra.rest import Client

KEY = YOUR_API_KEY
SECRET = YOUR_API_SECRET
PASSPHRASE = YOUR_API_PASSPHRASE

BTC_ACCOUNT_ID = YOUR_BTC_ACCOUNT_ID

loop = asyncio.get_event_loop()

async def get_btc_account():
    async with Client(loop, auth=True, key=KEY,
                      secret=SECRET, passphrase=PASSPHRASE) as client:

        btc_account = await client.account(BTC_ACCOUNT_ID)
        print(btc_account)

loop.run_until_complete(get_btc_account())
```

Running the above:

```
$ python3 btc_account_info.py
{'id': '1b121cbe-bd4-4c42-9e31-7047632fc7c7', 'currency': 'BTC', 'balance': '26.
↪1023109600000000', 'available': '26.09731096', 'hold': '0.0050000000000000',
↪'profile_id': '151d9abd-abcc-4597-ae40-b6286d72a0bd'}
```

3.2 WebSocket

While `copra.websocket.Client` is meant to be overridden, but it can be used 'as is' to test the module through the command line.

```
# btc_heartbeat.py

import asyncio

from copra.websocket import Channel, Client

loop = asyncio.get_event_loop()

ws = Client(loop, Channel('heartbeat', 'BTC-USD'))

try:
    loop.run_forever()
except KeyboardInterrupt:
    loop.run_until_complete(ws.close())
    loop.close()
```

Running the above:

```
$ python3 btc_heartbeat.py
{'type': 'subscriptions', 'channels': [{'name': 'heartbeat', 'product_ids': ['BTC-USD
↪']}]}
```

(continues on next page)

(continued from previous page)

```
{'type': 'heartbeat', 'last_trade_id': 45950713, 'product_id': 'BTC-USD', 'sequence':␣
↪6254273323, 'time': '2018-07-05T22:36:30.823000Z'}
{'type': 'heartbeat', 'last_trade_id': 45950714, 'product_id': 'BTC-USD', 'sequence':␣
↪6254273420, 'time': '2018-07-05T22:36:31.823000Z'}
{'type': 'heartbeat', 'last_trade_id': 45950715, 'product_id': 'BTC-USD', 'sequence':␣
↪6254273528, 'time': '2018-07-05T22:36:32.823000Z'}
{'type': 'heartbeat', 'last_trade_id': 45950715, 'product_id': 'BTC-USD', 'sequence':␣
↪6254273641, 'time': '2018-07-05T22:36:33.823000Z'}
{'type': 'heartbeat', 'last_trade_id': 45950715, 'product_id': 'BTC-USD', 'sequence':␣
↪6254273758, 'time': '2018-07-05T22:36:34.823000Z'}
{'type': 'heartbeat', 'last_trade_id': 45950720, 'product_id': 'BTC-USD', 'sequence':␣
↪6254273910, 'time': '2018-07-05T22:36:35.824000Z'}
.
.
.
```

A Coinbase Pro API key allows `copra.websocket.Client` to authenticate with the Coinbase WebSocket server giving you access to feeds specific to your user account.

```
# user_channel.py

import asyncio

from copra.websocket import Channel, Client

KEY = YOUR_API_KEY
SECRET = YOUR_API_SECRET
PASSPHRASE = YOUR_API_PASSPHRASE

loop = asyncio.get_event_loop()

channel = Channel('user', 'LTC-USD')

ws = Client(loop, channel, auth=True, key=KEY, secret=SECRET, passphrase=PASSPHRASE)

try:
    loop.run_forever()
except KeyboardInterrupt:
    loop.run_until_complete(ws.close())
    loop.close()
```

Running the above:

```
$ python3 user_channel.py
{'type': 'subscriptions', 'channels': [{'name': 'user', 'product_ids': ['LTC-USD']}]}
{'type': 'received', 'order_id': '42d2677d-0d37-435f-a776-e9e7f81ff22b', 'order_type
↪': 'limit', 'size': '50.00000000', 'price': '1.00000000', 'side': 'buy', 'client_oid
↪': '00098b59-4ac9-4ff8-ba16-bd2ef673f7b7', 'product_id': 'LTC-USD', 'sequence':␣
↪2311323871, 'user_id': '642394321fdf8242c4006432', 'profile_id': '039ee148-d490-
↪44f9-9aed-0d1f6412884', 'time': '2018-07-07T17:33:29.755000Z'}
{'type': 'open', 'side': 'buy', 'price': '1.00000000', 'order_id': '42d2677d-0d37-
↪435f-a776-e9e7f81ff22b', 'remaining_size': '50.00000000', 'product_id': 'LTC-USD',
↪'sequence': 2311323872, 'user_id': '642394321fdf8242c4006432', 'profile_id':
↪'039ee148-d490-44f9-9aed-0d1f6412884', 'time': '2018-07-07T17:33:29.755000Z'}
.
.
.
```

CHAPTER 4

Installation

4.1 Stable release

pip is the preferred method to install CoPrA, as it will always install the most recent stable release. To install CoPrA and its dependencies, run this command in your terminal application:

```
$ pip install copra
```

If you don't have pip installed, this Python installation guide can guide you through the process.

4.2 From sources

Alternatively, the source code for the CoPrA package can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/tpodlaski/copra
```

Or download the tarball:

```
$ curl  -OL https://github.com/tpodlaski/copra/tarball/master
```

Once you have a copy of the source, un-zipped and un-tarred if you downloaded the tarball, you can install it with:

```
$ python setup.py install
```

CHAPTER 5

REST

5.1 Usage

Warning: Any references made below to specific aspects of the Coinbase Pro API such as the data structures returned by methods may be out of date. Please visit Coinbase Pro's WebSocket REST API documentation for the authorative and up to date API information.

5.1.1 Introduction

copra.rest.Client, the asyncronous REST client class provided by CoPrA, is intentionally simple. Its methods were designed specifically to replicate all of the endpoints offered by the Coinbase Pro REST service, both in the parameters they expect and the data they return.

With very few exceptions there is a one to one correspondence between *copra.rest.Client* methods and the Coinbase endpoints. As often as possible, parameter names were kept the same and the json-encoded lists and dicts returned by the API server are, in turn, returned by the client methods untouched. This makes it simple to cross reference the CoPrA source code and documentation with Coinbase's API documentation.

Additionally, it should be relatively easy to extend the client in order to build finer grained ordering methods, sophisticated account management systems, and powerful market analytics tools.

5.1.2 Errors

While *copra.rest.Client* takes a hands-off approach to the data returned from API server, it does involve itself while preparing the user-supplied method paramters that will become part of the REST request. Specifically, in instances where the client can identify that the provided parameters will return an error from the API server, it raises a descriptive `ValueError` in order to avoid an unnecessary server call.

For example, the client method *copra.rest.Client.market_order()* has parameters for the amount of currency to purchase or sell, `size`, and for the amount of quote currency to use for the transaction, `funds`. For a market order, it is impossible to require both so if both are sent in the method call, the client raises a `ValueError`.

The *copra.rest.Client* API documentation details for each method in what instances `ValueErrors` are raised.

copra.rest.APIRequestError

On the other hand, there will be times the client cannot tell ahead of time that an API request will return an error. Insufficient funds, invalid account ids, improper authorization, and internal server errors are just a few examples of the errors a request may return.

Because there are many potential error types, and the Coinbase documentation does not list them all, the *copra.rest.Client* raises a generic error, `copra.rest.APIRequestError`, whenever the HTTP status code of an API server response is non-2xx.

The string representation of an `APIRequestError` is the message returned by the Coinbase server along the HTTP status code. `APIRequestError`, also has an additional field, `response`, is the aiohttp.ClientResponse object returned to the CoPrA client by the aiohttp request. This can be used to get more information about the request/response including full headers, etc. See the aiohttp.ClientResponse documentation to learn more about its attributes.

To get a feel for the types of errors that the Coinbase server may return, please see the Coinbase Pro API error documentation.

5.1.3 REST Client

The CoPrA REST client methods like their respective Coinbase REST API endpoints fall in one of two main categories: public and private. Public methods require no authentication. They offer access to market data and other publically avaiable information. The private methods *do* require authentication, specifically by means of an API key. To learn how to create an API key see the Coinbase Pro support article titled "How do I create an API key for Coinbase Pro?".

Initialization

`__init__(loop, url=URL, auth=False, key='', secret='', passphrase='')` [*API Documentation*]

Initialization of an unauthorized client only requires one parameter: the asyncio loop the client will be running in:

```
import asyncio

from copra.rest import Client

loop = asyncio.get_event_loop()

client = Client(loop)
```

To initialize an authorized client you will also need the key, secret, passphrase that Coinbase provides you when you request an API key:

```
import asyncio

from copra.rest import Client

loop = asyncio.get_event_loop()

client = Client(loop, auth=True, key=YOUR_KEY,
                secret=YOUR_SECRET, passphrase=YOUR_PASSPHRASE)
```

5.1.4 Client Lifecycle

The lifecycle of a long-lived client is straight forward:

```
client = Client(loop)

# Make a fortune trading Bitcoin here

await client.close()
```

Initialize the client, make as many requests as you need to, and then close the client to release any resources the underlying aiohttp session acquired. Note that the Python interpreter will complain if your program closes without closing your client first.

If you need to close the client from a function that is not a coroutine and the loop is remaining open you can close it like so:

```
loop.create_task(client.close())
```

Or, if the loop is closing, use:

```
loop.run_until_complete(client.close())
```

5.1.5 Context Manager

If you only need to create a client, use it briefly and not need it again for the duraction of your program, you can create it as context manager in which case the client is closed automatically when program execution leave the context manager block:

```
async with Client(loop) as client:
    client.do_something()
    client.do_something_else()
```

Note that if you will be using the client repeatedly over the duration of your program, it is best to create one client, store a reference to it, and use it repeatedly instead of creating a new client every time you need to make a request or two. This has to do with the aiohttp session handles its connection pool. Connections are reused and keep-alives are on which will result in better performance in subsequent requests versus creating a new client every time.

5.1.6 Public (Unauthenticated) Client Methods

Coinbase refers to the collection of endpoints that do not require authorization as their "Market Data API". They further group those endpoints into 3 categories: products, currency and time. The CoPra rest client provides methods that are a one-to-one match to the endpoints in Coinbase's Market Data API.

Products

- `products()` [*API Documentation*]
  Get a list of available currency pairs for trading.
- `order_book(product_id, level=1)` [*API Documentation*]
  Get a list of open orders for a product.
- `ticker(product_id)` [*API Documentation*]
  Get information about the last trade for a product.

- `trades(product_id, limit=100, before=None, after=None)` [*API Documentation*]
  List the latest trades for a product.
- `historic_rates(product_id, granularity=3600, start=None, stop=None)` [*API Documentation*]
  Get historic rates for a product.
- `get_24hour_stats(product_id)` [*API Documentation*]
  Get 24 hr stats for a product.

Currency

- `currencies()` [*API Documentation*]
  List known currencies.

Time

- `server_time` [*API Documentation*]
  Get the API server time.

5.1.7 Private (Authenticated) Client Methods

Coinbase labels its REST endpoints for account and order management as "private." Private in this sense means that they require authentication with the API server by signing all requests with a Coinbase API key. To use the corresponding `copra.rest.Client` methods you will need your own Coinbase API key. To learn how to create an API key see the Coinbase Pro support article titled "How do I create an API key for Coinbase Pro?"

Then you will need to initialize `copra.rest.Client` with that API key:

```
import asyncio

from copra.rest import Client

loop = asyncio.get_event_loop()

client = Client(loop, auth=True, key=YOUR_KEY,
                secret=YOUR_SECRET, passphrase=YOUR_PASSPHRASE)
```

Note: Even if you have created an authenticated client, it will only sign the requests to the Coinbase API server that require authentication. The "public" market data methods will still be made unsigned.

The Coinbase API documentation groups the "private" authenticated methods into these categories: accounts, orders, fills, deposits, withdrawals, stablecoin conversions, payment methods, Coinbase accounts, reports, and user account.

Again there is a one-to-one mapping from `copra.rest.Client` methods and their respective Coinbase API endpoints, but this time there is one exception. Coinbase has a single endpoint, "/orders" for placing orders. This enpoint handles both limit and market orders as well as the stop versions of both. Because of the number of parameters needed to cover all types of orders as well as the complicated interactions between the them, the decision was made to split this enpoint into two methods: *copra.rest.Client.limit_order()* and *copra.rest.Client.market_order()*.

Accounts

- `accounts()` [*API Documentation*]
  Get a list of your Coinbase Pro trading accounts.
- `account(account_id)` [*API Documentation*]
  Retrieve information for a single account.
- `account_history(account_id, limit=100, before=None, after=None)` [*API Documentation*]
  Retrieve a list account activity.
- `holds(account_id, limit=100, before=None, after=None)` [API Documentation]
  Get any existing holds on an account.

Orders

- `limit_order(side, product_id, price, size, time_in_force='GTC', cancel_after=None, post_only=False, client_oid=None, stp='dc',stop=None, stop_price=None)` [*API Documentation*]
  Place a limit order or a stop entry/loss limit order.
- `market_order(self, side, product_id, size=None, funds=None, client_oid=None, stp='dc', stop=None, stop_price=None)` [*API Documentation*]
  Place a market order or a stop entry/loss market order.
- `cancel(order_id)` [API Documentation]
  Cancel a previously placed order.
- `cancel_all(product_id=None, stop=False)` [API Documentation]
  Cancel "all" orders.
- `orders(status=None, product_id=None, limit=100, before=None, after=None)` [*API Documentation*]
  Retrieve a list orders
- `get_order(self, order_id)` [*API Documentation*]
  Get a single order by order id.

Fills

- `fills(order_id='', product_id='', limit=100, before=None, after=None)` [*API Documentation*]
  Get a list of recent fills.

Deposits

- `deposit_payment_method(amount, currency, payment_method_id)` [*API Documentation*]
  Deposit funds from a payment method on file.
- `deposit_coinbase(amount, currency, coinbase_account_id)` [*API Documentation*]
  Deposit funds from a Coinbase account.

Withdrawals

- `withdraw_payment_method(self, amount, currency, payment_method_id)` [*API Documentation*]
  Withdraw funds to a payment method on file.
- `withdraw_coinbase(amount, currency, coinbase_account_id)` [*API Documentation*]
  Withdraw funds to a Coinbase account.
- `withdraw_crypto(amount, currency, crypto_address)` [*API Documentation*]
  Withdraw funds to a crypto address.

Stablecoin Conversions

- `stablecoin_conversion(from_currency_id, to_currency_id, amount)` [API Documentation]
  Convert to and from a stablecoin.

Payment Methods

- `payment_methods()` [*API Documentation*]
  Get a list of the payment methods you have on file.

Fees

- `fees()` [*API Documentation*]
  Get your current maker & taker fee rates and 30-day trailing volume.

Coinbase Accounts

- `coinbase_accounts()` [*API Documentation*]
  Get a list of your coinbase accounts.

Reports

- `create_report(report_type, start_date, end_date, product_id='', account_id='', report_format='pdf', email='')` [*API Documentation*]
  Create a report about your account history.
- `report_status(report_id)` [*API Documentation*]
  Get the status of a report.

User Account

- `trailing_volume()` [*API Documentation*]
  Return your 30-day trailing volume for all products.

5.2 Examples

5.2.1 Market Order

The following example places a market order for the minimum amount of BTC-USD possible and then follows up by placing a stop loss market order at a stop price that is $300 less than the original order price.

This is a contrived example that makes some assumptions and does not count for every contingency, but it does use several `copra.rest.Client` methods. The steps that is follows are:

1. Check to see what the minimum BTC-USD order size is. *products()*
2. Check to see what the available USD balance is. *accounts()*
3. Check for the price of the last BTC-USD trade. *ticker()*
4. Place a market order. *market_order()*
5. Check the order status. *get_order()*
6. Place a stop loss market order. *market_order()*

```
# rest_example.py

import asyncio

from copra.rest import APIRequestError, Client

KEY = YOUR_KEY
SECRET = YOUR_SECRET
PASSPHRASE = YOUR_PASSPHRASE

loop = asyncio.get_event_loop()

async def run():

    async with Client(loop, auth=True, key=KEY, secret=SECRET,
                      passphrase=PASSPHRASE) as client:

        # Determine the smallest size order of BTC-USD possible.
        products = await client.products()
        for product in products:
            if product['id'] == 'BTC-USD':
                min_btc_size = float(product['base_min_size'])
                break

        print('\nMinimum BTC-USD order size: {}\n'.format(min_btc_size))

        # Get the amount of USD you have available. This assumes you don't
        # know the account id of your Coinbase Pro USD account. If you did,
        # you could just call client.account(account_id) to retrieve the
        # amount of USD available.
        accounts = await client.accounts()
        for account in accounts:
            if account['currency'] == 'USD':
                usd_available = float(account['available'])

        print('USD available: ${}\n'.format(usd_available))
```

(continues on next page)

(continued from previous page)

```
        # Get the last price of BTC-USD
        btc_usd_ticker = await client.ticker('BTC-USD')
        btc_usd_price = float(btc_usd_ticker['price'])

        print("Last BTC-USD price: ${}\n".format(btc_usd_price))

        # Verify you have enough USD to place the minimum BTC-USD order
        usd_needed = btc_usd_price * min_btc_size
        if usd_available < usd_needed:
            print('Sorry, you need ${} to place the minimum BTC order'.format(
                                                                    usd_needed))
            return

        # Place a market order for the minimum amount of BTC
        try:
            order = await client.market_order('buy', 'BTC-USD',
                                              size=min_btc_size)
            order_id = order['id']

            print('Market order placed.')
            print('\tOrder id: {}'.format(order_id))
            print('\tSize: {}\n'.format(order['size']))

        except APIRequestError as e:
            print(e)
            return

        # Wait a few seconds just to make sure the order completes.
        await asyncio.sleep(5)

        # Check the order status
        order = await client.get_order(order_id)

        # Assume the order is done and not rejected.
        order_size = float(order['filled_size'])
        order_executed_value = float(order['executed_value'])

        # We could check the fills to get the price(s) the order was
        # executed at, but we'll just use the average price.
        order_price = order_size * order_executed_value

        print('{} BTC bought at ${:.2f} for ${:.2f}\n'.format(order_size,
                                                              order_price,
                                                              order_executed_value))

        # Place a stop loss market order at $300 below the order price.
        stop_price = '{:.2f}'.format(6680.55 - 300)
        sl_order = await client.market_order('sell', 'BTC-USD', order_size,
                                             stop='loss', stop_price=stop_price)

        print('Stop loss order placed.')
        print('\tOrder id: {}'.format(sl_order['id']))
        print('\tSize: {}'.format(sl_order['size']))
        print('\tStop price: ${:.2f}\n'.format(float(sl_order['stop_price'])))

        await client.cancel_all(stop=True)
```

(continues on next page)

(continued from previous page)

```
loop.run_until_complete(run())

loop.close()
```

Running this script with your API key credentials inserted in their proper spots should yield output similar to that below.

```
$ python3 rest_example.py

Minimum BTC-USD order size: 0.001

USD available: $1485.6440517304

Last BTC-USD price: $6571.00

Market order placed.
        Order id: 1ed693ef-fc95-49ec-af6e-d37937d5ff1b
        Size: 0.00100000

0.001 BTC bought at $6571.00 for $6.57

Stop loss order placed.
        Order id: 72998d92-dea2-4f4c-83f2-e119f92861d5
        Size: 0.00100000
        Stop price: $6271.00

$
```

5.3 REST Public API Reference

The following is an API reference of CoPrA generated from Python source code and docstrings.

Warning: This is a *complete* reference of the *public* API of CoPrA. User code and applications should only rely on the public API, since internal APIs can (and will) change without any guarantees. Anything *not* listed here is considered a private API.

5.3.1 Module `copra.rest`

class copra.rest.**Client**(*loop*, *url='https://api.pro.coinbase.com'*, *auth=False*, *key=''*, *secret=''*, *passphrase=''*)

Asyncronous REST client for Coinbase Pro.

About Pagination

Most (but not all) client methods that return a list use cursor pagination. Cursor pagination allows for fetching results before and after the current page of results and is well suited for realtime data.

While non-paginated methods return a single value, paginated methods return a 3-tuple consisting of the page of results, the **before** cursor, and the **after** cursor.

The **before** cursor references the first item in a results page and the **after** cursor references the last item in a set of results.

To request a page of results *before* the current one (*newer chronologically*), use the **before** parameter. Your initial request can omit this parameter to get the default first page.

To request a page of results *after* the current one (*older chronologically*), use the **after** parameter.

__init__(*loop*, *url='https://api.pro.coinbase.com'*, *auth=False*, *key=''*, *secret=''*, *passphrase=''*)

Parameters

- **loop** (*asyncio loop*) – The asyncio loop that the client runs in.
- **url** (*str*) – (optional) The REST API server url. The default is https://api.pro.coinbase.com. This generally does not need to be changed. One reason to change it would be to test against the sandbox server.
- **auth** (*bool*) – (optional) Whether or not the (entire) REST session is authenticated. If True, you will need an API key from the Coinbase Pro website. The default is False.
- **key** (*str*) – (optional) The API key to use for authentication. Required if auth is True. The default is ''.
- **secret** (*str*) – (optional) The secret string for the API key used for authenticaiton. Required if auth is True. The default is ''.
- **passphrase** (*str*) – (optional) The passphrase for the API key used for authentication. Required if auth is True. The default is ''.

Raises **ValueError** – If auth is True and key, secret, and passphrase are not provided.

account(*account_id*)

Retrieve information for a single account.

Authorization

This method requires authorization. The API key must have either the "view" or "trade" permission.

Parameters **account_id** (*str*) – The account id.

Returns

A dict of account information.

Example:

```
{
  'id': 'a764610f-334e-4ece-b4dd-f31111ed58e7',
  'currency': 'USD',
  'balance': '1000.0000005931528000',
  'available': '1000.0000005931528',
  'hold': '0.0000000000000000',
  'profile_id': '019be148-d490-45f9-9ead-0d1f64127716'
}
```

Raises

- **ValueError** – If the client is not configured for authorization.
- **APIRequestError** – Any error generated by the Coinbase Pro API server.

account_history(*account_id*, *limit=100*, *before=None*, *after=None*)

Retrieve a list account activity.

Account activity includes transactions that either increase or decrease the account balance. Transactions are sorted latest first.

Authorization

This method requires authorization. The API key must have either the "view" or "trade" permission.

Pagination

This method is paginated. See *pagination* for more details.

Parameters

- **account_id** (*str*) – The account id.
- **limit** (*int*) – (optional) The number of results to be returned per request. The default (and maximum) value is 100.
- **before** (*int*) – (optional) The before cursor value. The default is None.
- **after** (*int*) – (optional) The after cursor value. The default is None.

Returns

A 3-tuple (results, before cursor, after cursor)

results is a list of dicts each representing an instance of account activity. The different types of activity returned are:

- **transfer** Funds moved to/from Coinbase to Coinbase Pro
- **match** Funds moved as a result of a trade
- **fee** Fee as a result of a trade
- **rebate** Fee rebate

The details field contains type-specific details about the specific transaction.

Example:

```
(
  [
    {
      'created_at': '2018-09-28T19:31:21.211159Z',
      'id': 10712040275,
      'amount': '-600.9103845810000000',
      'balance': '0.0000005931528000',
      'type': 'match',
      'details': {
                    'order_id': 'd2fadbb5-8769-4b80-91da-be3d9c6bd38d
↪',
                    'trade_id': '34209042',
                    'product_id': 'BTC-USD'
                  }
    },
    {
```

(continues on next page)

(continued from previous page)

```
      'created_at': '2018-09-23T23:13:45.771507Z',
      'id': 1065316993,
      'amount': '-170.0000000000000000',
      'balance': '6.7138918107528000',
      'type': 'transfer',
      'details': {
                    'transfer_id': 'd00841ff-c572-4726-b9bf-
↪17e783159256',
                    'transfer_type': 'withdraw'
                  }
    },
    ...
  ],
  '1071064024',
  '1008063508'
)
```

Raises

- **ValueError** –
  - The client is not configured for authorization.
  - before and after are both set.
- **APIRequestError** – Any error generated by the Coinbase Pro API server.

accounts()

Get a list of Coinbase Pro trading accounts.

Authorization

This method requires authorization. The API key must have either the "view" or "trade" permission.

Returns

A list of dicts where each dict contains information about a trading a account.

Example:

```
[
  {
    'id': 'a764610f-334e-4ece-b4dd-f31111ed58e7',
    'currency': 'USD',
    'balance': '1000.0000005931528000',
    'available': '1000.0000005931528',
    'hold': '0.0000000000000000',
    'profile_id': '019be148-d490-45f9-9ead-0d1f64127716'
  },
  ...
]
```

Raises

- **ValueError** – The client is not configured for authorization.
- **APIRequestError** – Any error generated by the Coinbase Pro API server.

cancel(*order_id*)
Cancel a previously placed order.

If the order had no matches during its lifetime its record may be purged. This means the order details will not be available with *copra.rest.Client.get_order()*.

Authorization

This method requires authorization. The API key must have the "trade" permission.

Parameters **order_id** (*str*) – The id of the order to be cancelled. This is the server-assigned order id and not the optional client_oid.

Returns

A list consisting of a single string entry, the id of the cancelled order.

Example:

```
["144c6f8e-713f-4682-8435-5280fbe8b2b4"]
```

Raises

- **ValueError** – The client is not configured for authorization.
- **APIRequestError** – Any error generated by the Coinbase Pro API server.

cancel_all(*product_id=None*, *stop=False*)
Cancel "all" orders.

The default behavior of this method (and the underlying Coinbase API method) is to cancel only *open* orders. Stop orders, once placed, have a status method of *active* and will not be cancelled.

Setting **stop** = True will also cancel stop orders or *copra.rest.Client.cancel()* can be used to cancel invidual stop orders.

Authorization

This method requires authorization. The API key must have the "trade" permission.

Parameters

- **product_id** (*str*) – (optional) Only cancel orders for the specified product. The default is None.
- **stop** (*bool*) – (optional) Also delete stop orders.

Returns

A list of the ids of orders that were successfully cancelled.

Example:

```
[
  "144c6f8e-713f-4682-8435-5lia280fbe8b2b4",
  "debe4907-95dc-442f-af3b-cec12f42ebda",
  "cf7aceee-7b08-4227-a76c-3858144323ab",
  "dfc5ae27-cadb-4c0c-beef-8994936fde8a",
```

(continues on next page)

(continued from previous page)

```
  "34fecfbf-de33-4273-b2c6-baf8e8948be4"
]
```

Raises

- **ValueError** – The client is not configured for authorization.
- **APIRequestError** – For any error generated by the Coinbase Pro API server.

close()
Close the client session and release all aquired resources.

closed
True if the client has been closed, False otherwise

coinbase_accounts()
Get a list of your coinbase accounts.

Authorization

This method requires authorization. The API key must have the "transfer" permission.

Returns

A list of dicts where each dict contains information about a Coinbase account.

Example:

```
[
    {
        "id": "fc3a8a57-7142-542d-8436-95a3d82e1622",
        "name": "ETH Wallet",
        "balance": "0.00000000",
        "currency": "ETH",
        "type": "wallet",
        "primary": false,
        "active": true
    },
    {
        "id": "2ae3354e-f1c3-5771-8a37-6228e9d239db",
        "name": "USD Wallet",
        "balance": "0.00",
        "currency": "USD",
        "type": "fiat",
        "primary": false,
        "active": true,
        "wire_deposit_information": {
            "account_number": "0199003122",
            "routing_number": "026013356",
            "bank_name": "Metropolitan Commercial Bank",
            "bank_address": "99 Park Ave 4th Fl New York, NY 10016",
            "bank_country": {
                "code": "US",
                "name": "United States"
            },
            "account_name": "Coinbase, Inc",
            "account_address": "548 Market Street, San Fran, CA␣
↪94104",
```

(continues on next page)

(continued from previous page)

```
            "reference": "BAOCAEUX"
        }
    },
    {
        "id": "1bfad868-5223-5d3c-8a22-b5ed371e55cb",
        "name": "BTC Wallet",
        "balance": "0.00000000",
        "currency": "BTC",
        "type": "wallet",
        "primary": true,
        "active": true
    },
    ...,
]
```

Raises

- **ValueError** – The client is not configured for authorization.
- **APIRequestError** – Any error generated by the Coinbase Pro API server.

create_report(*report_type*, *start_date*, *end_date*, *product_id=''*, *account_id=''*, *report_format='pdf'*, *email=''*)

Create a report about your account history.

Reports provide batches of historic information about your account in various human and machine readable forms.

The report will be generated when resources are available. Report status can be queried via *copra.rest.Client.report_status()*. The url for the report file will be available once the report has successfully been created and is available for download.

Note: Reports are only available for download for a few days after being created. Once a report expires, the report is no longer available for download and is deleted.

Authorization

This method requires authorization. The API key must have either the "view" or "trade" permission.

Parameters

- **report_type** (*str*) – The type of report to generate. This must be either "fills" or "account".
- **start_date** (*str*) – The starting date of the requested report as a str in ISO 8601 format.
- **end_date** (*str*) – The ending date of the requested report as a str in ISO 8601 format.
- **product_id** (*str*) – (optional) ID of the product to generate a fills report for. e.g. "BTC-USD". Required if type is fills.
- **account_id** (*str*) – (optional) ID of the account to generate an account report for. Required if type is account.

- **report_format** (*str*) – (optional) Format of the report to be generated. Can be either pdf or csv. The default is pdf.
- **email** (*str*) – (optional) Email address to send the report to. The default is None.

Returns

A dict of information about the report including its id which can be used to check its status.

Example:

```
{
    "id": "0428b97b-bec1-429e-a94c-59232926778d",
    "type": "fills",
    "status": "pending",
    "created_at": "2015-01-06T10:34:47.000Z",
    "completed_at": undefined,
    "expires_at": "2015-01-13T10:35:47.000Z",
    "file_url": undefined,
    "params": {
        "start_date": "2014-11-01T00:00:00.000Z",
        "end_date": "2014-11-30T23:59:59.000Z"
    }
}
```

Raises

- **ValueError** –
  - The client is not configured for authorization.
  - Invalid report_type provided.
  - report_type is fills and product_id is not provided.
  - report_type is account and account_id is not provided.
  - Invalid report_format provided.
- **APIRequestError** – Any error generated by the Coinbase Pro API server.

currencies()

List known currencies.

Currency codes will conform to the ISO 4217 standard where possible. Currencies which have or had no representation in ISO 4217 may use a custom code.

Note: Not all currencies may be currently in use for trading.

Returns

A list of dicts where each dict contains information about a currency.

Example:

```
[
  {
    "id": "BTC",
    "name": "Bitcoin",
    "min_size": "0.00000001"
  },
```

(continues on next page)

(continued from previous page)

```
  {
    "id": "USD",
    "name": "United States Dollar",
    "min_size": "0.01000000"
  },
  ...,
]
```

Raises **APIRequestError** – Any error generated by the Coinbase Pro API server.

delete (*path='/'*, *params=None*, *auth=False*)
Base method for making DELETE requests.

Parameters

- **path** (*str*) – (optional) The path not including the base URL of the resource to be deleted. The default is '/'.
- **params** (*dict*) – (optional) dict or MultiDict of key/value str pairs to be appended as the request's query string. The default is None.
- **auth** (*boolean*) – (optional) Indicates whether or not this request needs to be authenticated. The default is False.

Returns

A 2-tuple: (response headers, response body).

Response headers is a dict with the HTTP headers of the response. The response body is a JSON-formatted, UTF-8 encoded str.

Raises **APIRequestError** – Any error generated by the Coinbase Pro API server.

deposit_coinbase (*amount*, *currency*, *coinbase_account_id*)
Deposit funds from a Coinbase account.

Authorization

This method requires authorization. The API key must have the "transfer" permission.

Parameters

- **amount** (*float*) – The amount of the currency to deposit. This paramater may also be a string to avoid floating point issues.
- **currency** (*str*) – The type of currency to deposit. i.e., BTC, LTC, USD, etc.
- **coinbase_account_id** (*str*) – The id of the Coinbase account to deposit from. To get a list of Coinbase accounts, use: *copra.rest.Client.coinbase_accounts()*.

Returns

A dict with a deposit id and confirmation of the deposit amount and currency.

Example:

```
{
    "id": "593533d2-ff31-46e0-b22e-ca754147a96a",
    "amount": "10.00",
    "currency": "BTC",
}
```

Raises

- **ValueError** – The client is not configured for authorization.
- **APIRequestError** – Any error generated by the Coinbase Pro API server.

deposit_payment_method(*amount*, *currency*, *payment_method_id*)

Deposit funds from a payment method on file.

To get a list of available payment methods, use *copra.rest.Client.payment_methods()*.

Authorization

This method requires authorization. The API key must have the "transfer" permission.

Parameters

- **amount** (*float*) – The amount of the currency to deposit. This paramater may also be a string to avoid floating point issues.
- **currency** (*str*) – The type of currency to deposit. i.e., USD, EUR, etc.
- **payment_method_id** (*str*) – The id of the payment method to use.

Returns

A dict with a deposit id, timestamp and other deposit information.

Example:

```
{
    "id": "593533d2-ff31-46e0-b22e-ca754147a96a",
    "amount": "10.00",
    "currency": "USD",
    "payout_at": "2016-08-20T00:31:09Z"
}
```

Raises

- **ValueError** – The client is not configured for authorization.
- **APIRequestError** – Any error generated by the Coinbase Pro API server.

fees()

Get your current maker & taker fee rates and 30-day trailing volume.

Note: Quoted rates are subject to change. More information on fees can found at: https://support.pro.coinbase.com/customer/en/portal/articles/2945310-fees.

Returns

A dict containing the maker fee, taker fee and 30-day trailing volume.

Example:


```
{
    "maker_fee_rate": "0.0015",
    "taker_fee_rate": "0.0025",
    "usd_volume": "25000.00"
}
```


Raises **APIRequestError** – Any error generated by the Coinbase Pro API server.

fills(*order_id=''*, *product_id=''*, *limit=100*, *before=None*, *after=None*)

Get a list of recent fills.

Authorization

This method requires authorization. The API key must have either the "view" or "trade" permission.

Pagination

This method is paginated. See *pagination* for more details.

Parameters

- **order_id** (*str*) – (optional) Limit list of fills to this order_id, Either this or product_id must be defined.
- **product_id** (*str*) – (optional) Limit list of fills to this product_id. Either this or order_id must be defined.
- **limit** (*int*) – (optional) The number of results to be returned per request. The default (and maximum) value is 100.
- **before** (*int*) – (optional) The before cursor value. The default is None.
- **after** (*int*) – (optional) The after cursor value. The default is None.

Returns

A 3-tuple (fills, before cursor, after cursor)

fills is a list of dicts where each dict represents a fill.

Example:


```
[
  {
    "trade_id": 74,
    "product_id": "BTC-USD",
    "price": "10.00",
    "size": "0.01",
    "order_id": "d50ec984-77a8-460a-b958-66f114b0de9b",
    "created_at": "2014-11-07T22:19:28.578544Z",
    "liquidity": "T",
    "fee": "0.00025",
    "settled": true,
    "side": "buy"
  },
```


(continues on next page)

(continued from previous page)

```
  ...,
]
```

Raises

- **ValueError** –
  - The client is not configured for authorization.
  - Both before and after are set.
  - Neither order_id nor product_id are set or both are set.
- **APIRequestError** – Any error generated by the Coinbase Pro API server.

get (*path='/'*, *params=None*, *auth=False*)
Base method for making GET requests.

Parameters

- **path** (*str*) – (optional) The path not including the base URL of the resource to be retrieved. The default is '/'.
- **params** (*dict*) – (optional) dict or MultiDict of key/value str pairs to be appended as the request's query string. The default is None.
- **auth** (*boolean*) – (optional) Indicates whether or not this request needs to be authenticated. The default is False.

Returns

A 2-tuple: (response headers, response body).

Response headers is a dict with the HTTP headers of the response. The response body is a JSON-formatted, UTF-8 encoded str.

Raises **APIRequestError** – Any error generated by the Coinbase Pro API server.

get_24hour_stats (*product_id*)
Get 24 hr stats for a product.

Parameters **product_id** (*str*) – The product id.

Returns

A dict of stats for the product including: open, high, low, volume, last price, and 30 day volume.

Example:

```
{
  'open': '6710.37000000',
  'high': '6786.73000000',
  'low': '6452.02000000',
  'volume': '9627.98224214',
  'last': '6484.03000000',
  'volume_30day': '238376.24964395'
}
```

Raises **APIRequestError** – Any error generated by the Coinbase Pro API server.

get_order(*order_id*)
Get a single order by order id.

Authorization

This method requires authorization. The API key must have either the "view" or "trade" permission.

Parameters **order_id** (*str*) – The order id.

Returns

A dict of information about the order.

Example:

```
{
  "id": "68e6a28f-ae28-4788-8d4f-5ab4e5e5ae08",
  "size": "1.00000000",
  "product_id": "BTC-USD",
  "side": "buy",
  "stp": "dc",
  "funds": "9.9750623400000000",
  "specified_funds": "10.0000000000000000",
  "type": "market",
  "post_only": false,
  "created_at": "2016-12-08T20:09:05.508883Z",
  "done_at": "2016-12-08T20:09:05.527Z",
  "done_reason": "filled",
  "fill_fees": "0.0249376391550000",
  "filled_size": "0.01291771",
  "executed_value": "9.9750556620000000",
  "status": "done",
  "settled": true
}
```

Note: Open orders may change state between the request and the response depending on market conditions.

Raises

- **ValueError** – The client is not configured for authorization.
- **APIRequestError** – Any error generated by the Coinbase Pro API server.

historic_rates(*product_id*, *granularity=3600*, *start=None*, *end=None*)
Get historic rates for a product.

Rates are returned in grouped buckets based on the requested granularity.

Note: The maximum number of data points for a single request is 300 candles. If your selection of start/end time and granularity will result in more than 300 data points, your request will be rejected. If you wish to retrieve fine granularity data over a larger time range, you will need to make multiple requests with new start/end ranges.

Historical rate data may be incomplete. No data is published for intervals where there are no ticks.

Warning: Historical rates should not be polled frequently. If you need real-time information, use the trade and book endpoints along with the websocket feed.

Parameters

- **product_id** (*str*) – The product id.
- **granularity** (*int*) – (optional) Desired timeslice in seconds. The granularity field must be one of the following values: {60, 300, 900, 3600, 21600, 86400}. Otherwise, your request will be rejected. These values correspond to timeslices representing one minute, five minutes, fifteen minutes, one hour, six hours, and one day, respectively. The default is 3600 (1 hour).
- **start** (*str*) – (optional) The start time as a str in ISO 8601 format. This field is optional. If it is set, then end must be set as well. If neither start nor end are set, start will default to the time relative to now() that would return 300 results based on the granularity.
- **end** (*str*) – (optional) The end time as a str in ISO 8601 format. This field is optional. If it is set then start must be set as well. If it is not set, end will default to now().

Returns

A list of lists where each list item is a "bucket" representing a timeslice of length granularity. The fields of the bucket are: [time, low, high, open, close, volume]

- **time** - bucket start time as a Unix timestamp
- **low** - lowest price during the bucket interval
- **high** - highest price during the bucket interval
- **open** - opening price (first trade) in the bucket interval
- **close** - closing price (last trade) in the bucket interval
- **volume** - volume of trading activity during the bucket interval

Example:

```
[
  [1538179200, 61.12, 61.75, 61.74, 61.18, 2290.8172972700004],
  [1538175600, 61.62, 61.8, 61.65, 61.75, 2282.2335001199995],
  [1538172000, 61.52, 61.79, 61.66, 61.65, 3877.4680861400007],
  ...
]
```

Raises

- **ValueError** –
  - granularity is not one of the possible values.
  - Either start or end is set but not both
- **APIRequestError** – Any error generated by the Coinbase Pro API server.

holds(*account_id*, *limit=100*, *before=None*, *after=None*)

Get any existing holds on an account.

Holds are placed on an account for any active orders or pending withdraw requests. As an order is filled, the hold amount is updated. If an order is canceled, any remaining hold is removed. For a withdraw, once it is completed, the hold is removed.

Authorization

This method requires authorization. The API key must have either the "view" or "trade" permission.

Pagination

This method is paginated. See *pagination* for more details.

Parameters

- **account_id** (*str*) – The acount ID to be checked for holds.
- **limit** (*int*) – (optional) The number of results to be returned per request. The default (and maximum) value is 100.
- **before** (*int*) – (optional) The before cursor value. The default is None.
- **after** (*int*) – (optional) The after cursor value. The default is

Returns

A 3-tuple (holds, before cursor, after cursor)

holds is a list of dicts where each dict represents a hold on the account.

Example:

```
(
  [
    {
      "id": "82dcd140-c3c7-4507-8de4-2c529cd1a28f",
      "account_id": "e0b3f39a-183d-453e-b754-0c13e5bab0b3",
      "created_at": "2014-11-06T10:34:47.123456Z",
      "updated_at": "2014-11-06T10:40:47.123456Z",
      "amount": "4.23",
      "type": "order",
      "ref": "0a205de4-dd35-4370-a285-fe8fc375a273",
    },
    ...
  ],
  '1071064024',
  '1008063508'
)
```

Raises

- **ValueError** –
  - The client is not configured for authorization.
  - before and after are both set.
- **APIRequestError** – Any error generated by the Coinbase Pro API server.

limit_order(*side*, *product_id*, *price*, *size*, *time_in_force='GTC'*, *cancel_after=None*, *post_only=False*, *client_oid=None*, *stp='dc'*, *stop=None*, *stop_price=None*)

Place a limit order or a stop entry/loss limit order.

Authorization

This method requires authorization. The API key must have the "trade" permission.

Parameters

- **side** (*str*) – Either buy or sell
- **product_id** (*str*) – The product id to be bought or sold.
- **price** (*float*) – The price the order is to be executed at. This paramater may also be a string to avoid floating point issues.
- **size** (*float*) – The quantity of the cryptocurrency to buy or sell. This parameter may also be a string.
- **time_in_force** (*str*) – (optional) Time in force policies provide guarantees about the lifetime of an order. There are four policies: GTC (good till canceled), GTT (good till time), IOC (immediate or cancel), and FOK (fill or kill) GTT requires that cancel_after be set. IOC and FOK require post_only be False. The default is GTC.
- **cancel_after** (*str*) – (optional) The length of time before a GTT order is cancelled. Must be either min, hour, or day. time_in_force must be GTT or an error is raised. If cancel_after is not set for a GTT order, the order will be treated as GTC. The default is None.
- **post_only** (*bool*) – (optional) Indicates that the order should only make liquidity. If any part of the order results in taking liquidity, the order will be rejected and no part of it will execute. This flag is ignored for IOC and FOK orders. This value must be False for all stop orders. The default is False.
- **stp** (*str*) – (optional) Self trade preservation flag. The possible values are dc (decrease and cancel), co (cancel oldest), cn (cancel newest), or cb (cancel both). The default is dc.
- **stop** (*str*) – (optional) If this is a stop order, this value must be either loss or entry. Requires stop_price to be set. The default is None.
- **stop_price** (*float*) – (optioinal) The trigger price for stop orders. Required if stop is set. This may also be a string. The default is None.

Warning: As of 11/18, sending anything other than dc for stp while testing in Coinbase Pro's sandbox yields an APIRequestError "Invalid stp…" even though the Coinbase API documentation claims the other options for stp are valid. Change this from dc at your own risk.

Note: To see a more detailed explanation of these parameters and to learn more about the order life cycle, please see the official Coinbase Pro API documentation at: https://docs.gdax.com/#channels.

Returns

A dict of information about the order.

Example:


```
{
  'id': '97059421-3033-4cf4-99cb-925c1bf2c54f',
  'price': '35.00000000',
  'size': '0.00100000',
  'product_id': 'BTC-USD',
  'side': 'buy',
  'stp': 'dc',
  'type': 'limit',
  'time_in_force': 'GTC',
  'post_only': False,
  'created_at': '2018-11-25T21:24:37.166378Z',
  'fill_fees': '0.0000000000000000',
  'filled_size': '0.00000000',
  'executed_value': '0.0000000000000000',
  'status': 'pending',
  'settled': False
}
```


Raises

- **ValueError** –
  - The client is not configured for authorization.
  - The side is not either "buy" or "sell".
  - The time_in_force is not GTC, GTT, IOC or FOK.
  - time_in_force is GTT but cancel_after is not set
  - cancel_after for a limit order is set but isn't min, hour or day.
  - cancel_after is set for a limit order but time_in_force isn't GTT.
  - The time_in_force is IOC or FOK and post_only is True
  - stp is a value other than dc. co, cn, or cb.
  - stop is set to something other than loss or entry.
  - A stop order does not have stop_price set.
  - stop_price is set but stop is not
  - A stop_order has post_only set to True
- **APIRequestError** – Any error generated by the Coinbase Pro API server.

market_order(*side*, *product_id*, *size=None*, *funds=None*, *client_oid=None*, *stp='dc'*, *stop=None*, *stop_price=None*)

Place a market order or a stop entry/loss market order.

Authorization

This method requires authorization. The API key must have the "trade" permission.

Parameters

- **side** (*str*) – Either buy or sell
- **product_id** (*str*) – The product id to be bought or sold.

- **size** (*float*) – The quantity of the cryptocurrency to buy or sell. Either size or funds must be set for a market order but not both. This may also be a string. The default is None.
- **funds** (*float*) – This is the amount of quote currency to be used for a purchase (buy) or the amount to be obtained from a sale (sell). Either size or funds must be set for a market order but not both. This may also be a string. The default is None.
- **client_oid** (*str*) – (optional) A self generated UUID to identify the order. The default is None.
- **stp** (*str*) – (optional) Self trade preservation flag. The possible values are dc (decrease and cancel), co (cancel oldest), cn (cancel newest), or cb (cancel both). The default is dc.
- **stop** (*str*) – (optional) If this is a stop order, this value must be either loss or entry. Requires stop_price to be set. The default is None.
- **stop_price** (*float*) – (optional) The trigger price for stop orders. Required if stop is set. This may also be a string. The default is None.

Warning: As of 11/18, sending anything other than dc for stp while testing in Coinbase Pro's sandbox yields an APIRequestError "Invalid stp…" even though the Coinbase API documentation claims the other options for stp are valid. Change this from dc at your own risk.

Note: To see a more detailed explanation of these parameters and to learn more about the order life cycle, please see the official Coinbase Pro API documentation at: https://docs.gdax.com/#channels.

Returns

A dict of information about the order.

Example:

```
{
  'id': '37e81782-cc45-4ecc-a9ff-59c327bb1d40',
  'size': '0.00100000',
  'product_id': 'BTC-USD',
  'side': 'sell',
  'stp': 'dc',
  'type': 'market',
  'post_only': False,
  'created_at': '2018-11-25T21:28:04.788042Z',
  'fill_fees': '0.0000000000000000',
  'filled_size': '0.00000000',
  'executed_value': '0.0000000000000000',
  'status': 'pending',
  'settled': False
}
```

Raises

- **ValueError** –
  - The client is not configured for authorization.
  - The side is not either "buy" or "sell".
  - Neither size nor funds is set.

- Both size and funds are set
- stp is a value other than dc. co, cn, or cb.
- stop is set to something other than loss or entry.
- A stop order does not have stop_price set.
- stop_price is set but stop is not

- **APIRequestError** – Any error generated by the Coinbase Pro API server.

order_book(*product_id*, *level=1*)

Get a list of open orders for a product.

By default, only the inside (i.e. best) bid and ask are returned. This is equivalent to a book depth of 1 level. If you would like to see a larger order book, specify the level query parameter.

Level	Description
1	Only the best bid and ask
2	Top 50 bids and asks (aggregated)
3	Full order book (non aggregated)

If a level is not aggregated, then all of the orders at each price will be returned. Aggregated levels return only one size for each active price (as if there was only a single order for that size at the level).

Levels 1 and 2 are aggregated. The first field is the price. The second is the size which is the sum of the size of the orders at that price, and the third is the number of orders, the count of orders at that price. The size should not be multiplied by the number of orders.

Level 3 is non-aggregated and returns the entire order book.

Note: This request is NOT paginated. The entire book is returned in one response.

Level 1 and Level 2 are recommended for polling. For the most up-to-date data, consider using the websocket stream.

Warning: Level 3 is only recommended for users wishing to maintain a full real-time order book using the websocket stream. Abuse of Level 3 via polling will cause your access to be limited or blocked.

Parameters

- **product_id** (*str*) – The product id of the order book.
- **level** (*int*) – (optional) The level customizes the amount of detail shown. See bove for more detail. The default is 1.

Returns

A dict representing the order book for the product id specified. The layout of the dict will vary based on the level. See the examples below.

Level 1:

```
{
  'sequence': 7068939079,
  'bids': [['482.98', '54.49144003', 18]],
  'asks': [['482.99', '4.57036219', 10]]
}
```

Level 2:

```
{
  'sequence': 7069016926,
  'bids': [['489.13', '0.001', 1], ['487.99', '0.03', 1], ...],
  'asks': [['489.14', '40.72125158', 16], ['490.11', '0.5', 1], ...
↪],
}
```

Level 3:

```
{
  'sequence': 7072737439,
  'bids':
    [
      ['468.9', '0.01100413', '48c3ed25-616d-430d-bab4-cb338b489a33
↪'],
      ['468.9', '0.224', 'b96424ea-e992-4df5-b503-df50dac1ac50'],
      ...
    ],
  'asks':
    [
      ['468.91', '5.96606527', 'cc37e457-020c-4843-9a3e-e6164dcf4e60
↪'],
      ['468.91', '0.00341509', '43e8158a-30c6-437b-9a51-9b9da00e4e22
↪'],
      ...
    ]
}
```

Raises

- **ValueError** – level not 1, 2, or 3.
- **APIRequestError** – Any error generated by the Coinbase Pro API server.

orders (*status=None*, *product_id=None*, *limit=100*, *before=None*, *after=None*)

Retrieve a list orders.

The status of an order may change between the request and response depending on market conditions.

Authorization

This method requires authorization. The API key must have either the "view" or "trade" permission.

Pagination

This method is paginated. See *pagination* for more details.

Parameters

- **status** (*str*) – (optional) Limit list of orders to one or more of these statuses: open, pending, active or all. status may be a single string or a list of strings. i.e, [‘open’, ‘active’]. ‘all’ returns orders of all statuses. The default is [‘open’, ‘active’, ‘pending’].
- **product_id** (*str*) – (optional) Filter orders by product_id
- **limit** (*int*) – (optional) The number of results to be returned per request. The default (and maximum) value is 100.
- **before** (*int*) – (optional) The before cursor value. The default is None.
- **after** (*int*) – (optional) The after cursor value. The default is None.

Returns

A 3-tuple: (orders, before cursor, after cursor)

orders is a list of dicts where each dict represents an order.

Example:


```
([
  {
    "id": "d0c5340b-6d6c-49d9-b567-48c4bfca13d2",
    "price": "0.10000000",
    "size": "0.01000000",
    "product_id": "BTC-USD",
    "side": "buy",
    "stp": "dc",
    "type": "limit",
    "time_in_force": "GTC",
    "post_only": false,
    "created_at": "2016-12-08T20:02:28.53864Z",
    "fill_fees": "0.0000000000000000",
    "filled_size": "0.00000000",
    "executed_value": "0.0000000000000000",
    "status": "open",
    "settled": false
  },
  {
    "id": "8b99b139-58f2-4ab2-8e7a-c11c846e3022",
    "price": "1.00000000",
    "size": "1.00000000",
    "product_id": "BTC-USD",
    "side": "buy",
    "stp": "dc",
    "type": "limit",
    "time_in_force": "GTC",
    "post_only": false,
    "created_at": "2016-12-08T20:01:19.038644Z",
    "fill_fees": "0.0000000000000000",
    "filled_size": "0.00000000",
    "executed_value": "0.0000000000000000",
    "status": "open",
    "settled": false
  }
 ],
 '1071064024',
 '1008063508'
)
```

Raises

- **ValueError** –
  - The client is not configured for authorization.
  - An invalid status string is provided.
  - before and after are both set.
- **APIRequestError** – Any error generated by the Coinbase Pro API server.

payment_methods()

Get a list of the payment methods you have on file.

Authorization

This method requires authorization. The API key must have the "transfer" permission.

Returns

A list of dicts where each dict contains detailed information about a payment method the account has available.

Example:


```
[
    {
        "id": "bc6d7162-d984-5ffa-963c-a493b1c1370b",
        "type": "ach_bank_account",
        "name": "Bank of America - eBan... ********7134",
        "currency": "USD",
        "primary_buy": true,
        "primary_sell": true,
        "allow_buy": true,
        "allow_sell": true,
        "allow_deposit": true,
        "allow_withdraw": true,
        "limits": {
            "buy": [
                {
                    "period_in_days": 1,
                    "total": {
                        "amount": "10000.00",
                        "currency": "USD"
                    },
                    "remaining": {
                        "amount": "10000.00",
                        "currency": "USD"
                    }
                }
            ],
            "instant_buy": [
                {
                    "period_in_days": 7,
                    "total": {
                        "amount": "0.00",
                        "currency": "USD"
                    },
```


(continues on next page)

(continued from previous page)


```
                    "remaining": {
                        "amount": "0.00",
                        "currency": "USD"
                    }
                }
            ],
            "sell": [
                {
                    "period_in_days": 1,
                    "total": {
                        "amount": "10000.00",
                        "currency": "USD"
                    },
                    "remaining": {
                        "amount": "10000.00",
                        "currency": "USD"
                    }
                }
            ],
            "deposit": [
                {
                    "period_in_days": 1,
                    "total": {
                        "amount": "10000.00",
                        "currency": "USD"
                    },
                    "remaining": {
                        "amount": "10000.00",
                        "currency": "USD"
                    }
                }
            ]
        }
    },
]
```


Raises

- **ValueError** – The client is not configured for authorization.
- **APIRequestError** – Any error generated by the Coinbase Pro API server.

post(*path='/'*, *data=None*, *auth=False*)

Base method for making POST requests.

Parameters

- **path** (*str*) – (optional) The path not including the base URL of the resource to be POST'ed to. The default is '/'
- **data** (*dict*) – (optional) Dictionary of key/value str pairs to be sent in the body of the request. The default is None.
- **auth** (*boolean*) – (optional) Indicates whether or not this request needs to be authenticated. The default is False.

Returns

A 2-tuple: (response headers, response body).

Response headers is a dict with the HTTP headers of the response. The response body is a JSON-formatted, UTF-8 encoded str.

Raises **APIRequestError** – Any error generated by the Coinbase Pro API server.

products()

Get a list of available currency pairs for trading.

The base_min_size and base_max_size fields define the min and max order size.

The quote_increment field specifies the min order price as well as the price increment. The order price must be a multiple of this increment (i.e. if the increment is 0.01, order prices of 0.001 or 0.021 would be rejected).

Note: Product ID will not change once assigned to a product but the min/max/quote sizes can be updated in the future.

Returns

A list of dicts representing the currency pairs available for trading.

Example:

```
[
  {
    'id': 'BTC-USD',
    'base_currency': 'BTC',
    'quote_currency': 'USD',
    'base_min_size': '0.001',
    'base_max_size': '70',
    'quote_increment': '0.01',
    'display_name': 'BTC/USD',
    'status': 'online',
    'margin_enabled': False,
    'status_message': None,
    'min_market_funds': '10',
    'max_market_funds': '1000000',
    'post_only': False,
    'limit_only': False,
    'cancel_only': False
  },
  ...
 ]
```

Raises **APIRequestError** – Any error generated by the Coinbase Pro API server.

report_status(*report_id*)

Get the status of a report.

Once a report request has been accepted for processing, the status is available by polling the report resource endpoint.

The possible status values are:

- **pending** - The report request has been accepted and is awaiting processing.
- **creating** - The report is being created.
- **ready** - The report is ready for download from file_url.

The final report will be uploaded and available at file_url once the status indicates ready.

Authorization

This method requires authorization. The API key must have either the "view" or "trade" permission.

Parameters **report_id** (*str*) – The id of the report. This is obtained from *copra.rest.Client.create_report()*.

Returns

A dict summarizing the current status of the report. Examples follow.

Creating report:

```
{
    "id": "0428b97b-bec1-429e-a94c-59232926778d",
    "type": "fills",
    "status": "creating",
    "created_at": "2015-01-06T10:34:47.000Z",
    "completed_at": undefined,
    "expires_at": "2015-01-13T10:35:47.000Z",
    "file_url": undefined,
    "params": {
        "start_date": "2014-11-01T00:00:00.000Z",
        "end_date": "2014-11-30T23:59:59.000Z"
    }
}
```

Finished report:

```
{
    "id": "0428b97b-bec1-429e-a94c-59232926778d",
    "type": "fills",
    "status": "ready",
    "created_at": "2015-01-06T10:34:47.000Z",
    "completed_at": "2015-01-06T10:35:47.000Z",
    "expires_at": "2015-01-13T10:35:47.000Z",
    "file_url": "https://example.com/0428b97b.../fills.pdf",
    "params": {
        "start_date": "2014-11-01T00:00:00.000Z",
        "end_date": "2014-11-30T23:59:59.000Z"
    }
}
```

Raises

- **ValueError** – The client is not configured for authorization.
- **APIRequestError** – Any error generated by the Coinbase Pro API server.

server_time()

Get the API server time.

Returns

A dict with two fields: iso and epoch. iso is an ISO 8601 str, and epoch is a float. Both represent the current time at the API server.

Example:

```
{
  'iso': '2018-09-29T03:02:27.753Z',
  'epoch': 1538190147.753
}
```

Raises **APIRequestError** – Any error generated by the Coinbase Pro API server.

stablecoin_conversion (*from_currency_id*, *to_currency_id*, *amount*)
Convert to and from a stablecoin.

Authorization

This method requires authorization. The API key must have the "trade" permission.

Note: As of November, 1018, Coinbase Pro only supports USD-USDC conversions

Parameters

- **from_currency_id** (*str*) – The id of the currency to convert from.
- **to_currency_id** (*str*) – The id of the currency to convert to.
- **amount** (*float*) – The amount of currency to convert. This paramater may also be a string to avoid floating point issues.

Returns

A dict summarizing the conversion.

Example:

```
{
    "id": "8942caee-f9d5-4600-a894-4811268545db",
    "amount": "10000.00",
    "from_account_id": "7849cc79-8b01-4793-9345-bc6b5f08acce",
    "to_account_id": "105c3e58-0898-4106-8283-dc5781cda07b",
    "from": "USD",
    "to": "USDC"
}
```

Raises

- **ValueError** – The client is not configured for authorization.
- **APIRequestError** – Any error generated by the Coinbase Pro API server.

ticker (*product_id*)
Get information about the last trade for a specific product.

Note: Polling is discouraged in favor of connecting via the websocket stream and listening for match messages.

Parameters **product_id** (*str*) – The product id of the tick to be retrieved.

Returns

A dict containing information about the last trade (tick) for the product.

Example:

```
{
  'trade_id': 51554088,
  'price': '6503.14000000',
  'size': '0.00532605',
  'bid': '6503.13',
  'ask': '6503.14',
  'volume': '6060.89272148',
  'time': '2018-09-27T13:18:42.571000Z'
}
```

Raises **APIRequestError** – Any error generated by the Coinbase Pro API server.

trades(*product_id*, *limit=100*, *before=None*, *after=None*)

List the latest trades for a product.

The trade side indicates the maker order side. The maker order is the order that was open on the order book. buy side indicates a down-tick because the maker was a buy order and their order was removed. Conversely, sell side indicates an up-tick.

Pagination

This method is paginated. See *pagination* for more details.

Parameters

- **product_id** (*str*) – The product id whose trades are to be retrieved.
- **limit** (*int*) – (optional) The number of results to be returned per request. The default (and maximum) value is 100.
- **before** (*int*) – (optional) The before cursor value. The default is None.
- **after** (*int*) – (optional) The after cursor value. The default is None.

Returns

A 3-tuple: (trades, before cursor, after cursor)

trades is a list of dicts representing trades for the product specified.

Example:

```
(
  [
    {
      'time': '2018-09-27T22:49:16.105Z',
      'trade_id': 51584925,
      'price': '6681.01000000',
      'size': '0.02350019',
      'side': 'sell'
    },
    {
      'time': '2018-09-27T22:49:12.39Z',
      'trade_id': 51584924,
```

(continues on next page)

(continued from previous page)

```
      'price': '6681.00000000',
      'size': '0.01020000',
      'side': 'buy'
    },
    ...
  ],
  '51590012',
  '51590010'
)
```

Raises

- **ValueError** – before and after paramters are both provided.
- **APIRequestError** – Any error generated by the Coinbase Pro API server.

trailing_volume()

Return your 30-day trailing volume for all products.

This is a cached value that's calculated every day at midnight UTC.

Authorization

This method requires authorization. The API key must have either the "view" or "trade" permission.

Returns

A list of dicts where each dict contains information about a specific product that was traded.

Example:

```
[
    {
        "product_id": "BTC-USD",
        "exchange_volume": "11800.00000000",
        "volume": "100.00000000",
        "recorded_at": "1973-11-29T00:05:01.123456Z"
    },
    {
        "product_id": "LTC-USD",
        "exchange_volume": "51010.04100000",
        "volume": "2010.04100000",
        "recorded_at": "1973-11-29T00:05:02.123456Z"
    }
]
```

Raises

- **ValueError** – The client is not configured for authorization.
- **APIRequestError** – Any error generated by the Coinbase Pro API server.

withdraw_coinbase(*amount*, *currency*, *coinbase_account_id*)

Withdraw funds to a coinbase account.

Authorization

This method requires authorization. The API key must have the "transfer" permission.

Parameters

- **amount** (*float*) – The amount of the currency to withdraw. This paramater may also be a string to avoid floating point issues.
- **currency** (*str*) – The type of currency to withdraw. i.e., BTC, LTC, USD, etc.
- **coinbase_account_id** (*str*) – The id of the Coinbase account to withdraw to. To get a list of Coinbase accounts, use: *copra.rest.Client.coinbase_accounts()*.

Returns

A dict with the withdrawal id, and confirmation of the withdrawl amount and currency.

Example:

```
{
    "id":"593533d2-ff31-46e0-b22e-ca754147a96a",
    "amount":"10.00",
    "currency": "BTC",
}
```

Raises

- **ValueError** – The client is not configured for authorization.
- **APIRequestError** – Any error generated by the Coinbase Pro API server.

withdraw_crypto(*amount*, *currency*, *crypto_address*)

Withdraw funds to a crypto address.

Authorization

This method requires authorization. The API key must have the "transfer" permission.

Parameters

- **amount** (*float*) – The amount of the currency to withdraw. This paramater may also be a string to avoid floating point issues.
- **currency** (*str*) – The type of currency to withdraw. i.e., BTC, LTC, USD, etc.
- **crypto_address** (*str*) – The crypto address of the recipient.

Returns

A dict with the withrawal id and confirmation of the withdrawl amount and currency.

Example:

```
{
    "id":"593533d2-ff31-46e0-b22e-ca754147a96a",
    "amount":"10.00",
    "currency": "BTC",
}
```

Raises

- **ValueError** – The client is not configured for authorization.
- **APIRequestError** – Any error generated by the Coinbase Pro API server.

withdraw_payment_method(*amount*, *currency*, *payment_method_id*)
Withdraw funds to a payment method on file.

To get a list of available payment methods, use *copra.rest.Client.payment_methods()*.

Authorization

This method requires authorization. The API key must have the "transfer" permission.

Parameters

- **amount** (*float*) – The amount of the currency to withdraw. This paramater may also be a string to avoid floating point issues.
- **currency** (*str*) – The type of currency to withdraw. i.e., USD, EUR, etc.
- **payment_method_id** (*str*) – The id of the payment method on file to use.

Returns

A dict with a withdrawal id, timestamp, and other withdrawal information.

Example:

```
{
    "id":"593533d2-ff31-46e0-b22e-ca754147a96a",
    "amount": "10.00",
    "currency": "USD",
    "payout_at": "2016-08-20T00:31:09Z"
}
```

Raises

- **ValueError** – The client is not configured for authorization.
- **APIRequestError** – Any error generated by the Coinbase Pro API server.

CHAPTER 6

WebSocket

6.1 Usage

Warning: Any references made below to specific aspects of the Coinbase Pro API such as the channels and the data they provide may be out of date. Please visit Coinbase Pro's WebSocket API documentation for the authorative and up to date information.

6.1.1 Introduction

The CoPrA API provides two classes for creating a WebSocket client for the Coinbase Pro platform. The first, `copra.websocket.Channel`, is intended to be used "as is." The second, `copra.websocket.Client`, is the actual client class. It provides multiple callback methods to manage every stage of the client's life cycle.

6.1.2 Channel

At the heart of every WebSocket connection is the concept of a channel. A channel provides a specific type of data about one or more currency pairs. `copra.websocket.Channel` has two attributes: it's name `name` and the product pairs the channel is observing, `product_ids`.

The current channels provided by the Coinbase Pro API are:

- **heartbeart** - heartbeat messages are generated once a second. They include sequence numbers and last trade IDs that can be used to verify no messages were missed.
- **ticker** - ticker messages are sent every time a match happens providing real-time price updates.
- **level2** - level2 messages provide a high level view of the order book. After the initial snapshot of the order book is delivered, messages are sent every time the volume at specific price tier on the buy or sell side changes.
- **full** - the full channel provides real-time updates on orders and trades. There are messages for every stage of an orders life cycle including: received, open, match, done, change, and activate.

- **user** - the user channel provides the same information as the full channel but only for the authenticated user. As such you will need to be authenticated to susbsribe. This requires a Coinbase Pro API key.
- **matches** - this channel consists only of the match messages from the full channel.

The Coinbase Pro exchange currently hosts four digital currencies:

- **BTC** - Bitcoin
- **BCH** - Bitcoin Cash
- **ETH** - Etherium
- **LTC** - Litecoin Cash

And allows 3 fiat currencies for trading:

- **USD** - US Dollar
- **EUR** - Euro
- **GBP** - Great British Pounds (Sterling)

Not every combination of currencies is available for trading, however. The current currency pairs (or products) avaialable for trade are:

- **BTC-USD**
- **BTC-EUR**
- **BTC-GBP**
- **ETH-USD**
- **ETH-EUR**
- **ETH-BTC**
- **LTC-USD**
- **LTC-EUR**
- **LTC-BTC**
- **BCH-USD**
- **BCH-EUR**
- **BCH-BTC**

These are the product IDs referenced below.

Before connecting to the Coinbase Pro Websocket server, you will need to create one or more channels to subscribe to.

First, import the `Channel` class:

```
from copra.websocket import Channel
```

The channel is then initialized with its name and one or more product IDs. The heartbeat channel for the Bitcoin/US dollar pair would be initialized:

```
channel = Channel('heartbeat', 'BTC-USD')
```

A channel that recieves ticker information about the pairs Etherium/US dollar and Litecoin/Euro would be initialized:

```
channel = Channel('ticker', ['ETH-USD', 'LTC-EUR'])
```

As illustrated above, the product ID argument to the `Channel` constructor can be a single string or a list of strings.

To listen for messages about Bitcoin/US Dollar and Litecoin/Bitcoin orders for an authenticated user:

```
channel = Channel('user', ['BTC-USD', 'LTC-BTC'])
```

As noted above, this will require that the `Client` be authenticated. This is covered below.

6.1.3 Client

The `Client` class represents the Coinbase Pro WebSocket client. While it can be used "as is", most developers will want to subclass it in order to customize the behavior of its callback methods.

First it needs to be imported:

```
from copra.websocket import Client
```

For reference, the signature of the `Client` `__init__` method is:

```
def __init__(self, loop, channels, feed_url=FEED_URL,
             auth=False, key='', secret='', passphrase='',
             auto_connect=True, auto_reconnect=True,
             name='WebSocket Client')
```

Only two parameters are required to create a client: `loop` and `channels`.

`loop` is the Python asyncio loop that the client will run in. Somewhere in your code you will likely have something like:

```
import asyncio

loop = asyncio.get_event_loop()
```

`channels` is either a single `Channel` or a list of `Channels` the client should immediately subscribe to.

`feed_url` is the url of the Coinbase Pro Websocket server. The default is `copra.websocket.FEED_URL` which is wss://ws-feed.pro.coinbase.com:443.

If you want to test your code in Coinbase's "sandbox" development environment, you can set `feed_url` to `copra.websocket.SANDBOX_FEED_URL` which is wss://ws-feed-public.sandbox.pro.coinbase.com:443.

`auth` indicates whether or not the client will be authenticated. If True, you will need to also provide `key`, `secret`, and `passphrase`. These values are provided by Coinbase Pro when you register for an API key.

`auto_connect` determines whether or not to automatically add the client to the asyncio loop. If true, the client will be added to the loop when it (the client) is initialized. If the loop is already running, the WebSocket connection will open. If the loop is not yet running, the connection will be made as soon as the loop is started.

If `auto_connect` is False, you will need to explicitly call `client.add_as_task_to_loop()` when you are ready to add the client to the asyncio loop and open the WebSocket connection.

`auto_reconnect` determines the client's behavior is the connection is closed in any way other than by explicitly calling its `close` method. If True, the client will automatically try to reconnect and re-subscribe to the channels it subscribed to when the connection unexpectedly closed.

`name` is a simple string representing the name of the client. Setting this to something unique may be useful for logging purposes.

Callback Methods

The `Client` class provides four methods that are automatically called at different stages of the client's life cycle. The method that will be most useful for developers is `on_message()`.

on_open()

`on_open` is called as soon as the initial WebSocket opening handshake is complete. The connection is open, but the client is **not yet subscribed**.

If you override this method it is important that **you still call it** from your subclass' `on_open` method, since the parent method sends the initial subscription request to the WebSocket server. Somewhere in your `on_open` method you should have `super().on_open()`.

In addition to sending the subsciption request, this method also logs that the connection was opened.

on_message(message)

`on_message` is called everytime a message is received. `message` is a dict representing the message. Its content will depend on the type of message, the channels subscribed to, etc. Please read Coinbase Pro's WebSocket API documentation to learn about these message formats.

Note that with the exception of errors, every other message triggers this method including things like subscription confirmations. Your code should be prepared to handle unexpected messages.

This default method just prints the message received. If you override this method, there is no need to call the parent method from your subclass' method.

on_error(message, reason)

`on_error` is called when an error message is received from the WebSocket server. `message` a is string representing the error, and `reason` is a string that provides additional information about the cause of the error. Note that in many cases `reason` is blank.

The default implementation just logs the message and reason. If you override this method, your subclass only needs to call the parent's method if want to preserve this logging behavior.

on_close(was_clean, code, reason)

`on_close` is called whenever the connection between the client and server is closed. `was_clean` is a boolean indicating whether or not the connection was cleanly closed. `code`, an integer, and `reason`, a string, are sent by the end that initiated closing the connection.

If the client did not initiate this closure and `client.auto_reconnect` is set to True, the client will attempt to reconnect to the server and resubscribe to the channels it was subscribed to when the connection was closed. This method also logs the closure.

If your subclass overrides this method, it is important that the subclass method calls the parent method if you want to preserve the auto reconnect functionality. This can be done by including `super().on_close(was_clean, code, reason)` in your subclass method.

Other Methods

close()

`close` is called to close the connection to the WebSocket server. Note that if you call this method, the client will not attempt to auto reconnect regardless of what the value of `client.auto_reconnect` is.

subscribe(channels)

subscribe is called to susbcribe to additional channels. `channels` is either a single Channel or a list of Channels.

The original channels to be subscribed to are defined during the client's initialization. `subscribe` can be used to add channels whether the client has been added to asyncio loop yet or not. If the loop isn't yet running, the client will subscribe to all of its channels when it is. If the loop is already running, the subcription will be appended with new channels, and incoming data will be immediately received.

unsubscribe(channels)

`unsubscribe` is called to unsubscribe from channels. `channels` is either a single Channel or a list of Channels.

Like `subscribe`, `unsubscribe` can be called regardless of whether or not the client has already been added to the asyncio loop. If the client has not yet been added, `unsubscribe` will remove those channels from the set of channels to be initially subscribed to. If the client has already been added to the loop, `unsubscribe` will remove those channels from the subscription, and data flow from them will stop immediately.

6.2 Examples

6.2.1 Ticker

The following code, saved as `ticker.py`, when run from the command line prints a running ticker for the product ID supplied as an argument to the script.

```
#!/usr/bin/env python3

import asyncio
from datetime import datetime
import sys

from copra.websocket import Channel, Client

class Tick:

    def __init__(self, tick_dict):
        self.product_id = tick_dict['product_id']
        self.best_bid = float(tick_dict['best_bid'])
        self.best_ask = float(tick_dict['best_ask'])
        self.price = float(tick_dict['price'])
        self.side = tick_dict['side']
        self.size = float(tick_dict['last_size'])
        self.time = datetime.strptime(tick_dict['time'], '%Y-%m-%dT%H:%M:%S.%fZ')

    @property
```

(continues on next page)

(continued from previous page)

```
    def spread(self):
        return self.best_ask - self.best_bid

    def __repr__(self):

        rep  = "{}\t\t\t\t   {}\n".format(self.product_id, self.time)
        rep += "============================================================\n"
        rep += "   Price: ${:.2f}\t    Size: {:.8f}\t  Side: {: >5}\n".format(self.
↪price, self.size, self.side)
        rep += "Best ask: ${:.2f}\tBest bid: ${:.2f}\tSpread: ${:.2f}\n".format(self.
↪best_ask, self.best_bid, self.spread)
        rep += "============================================================\n"
        return rep


class Ticker(Client):

    def on_message(self, message):
        if message['type'] == 'ticker' and 'time' in message:
            tick = Tick(message)
            print(tick, "\n\n")

product_id = sys.argv[1]

loop = asyncio.get_event_loop()

channel = Channel('ticker', product_id)

ticker = Ticker(loop, channel)

try:
    loop.run_forever()
except KeyboardInterrupt:
    loop.run_until_complete(ticker.close())
    loop.close()
```

Streaming a ticker for LTC-USD:

```
$ ./ticker.py LTC-USD

LTC-USD                              2018-07-12 21:40:38.501000
============================================================
   Price: $75.73            Size: 0.22134981      Side:   buy
Best ask: $75.73        Best bid: $75.67        Spread: $0.06
============================================================



LTC-USD                              2018-07-12 21:40:38.501000
============================================================
   Price: $75.74            Size: 0.29362708      Side:   buy
Best ask: $75.74        Best bid: $75.67        Spread: $0.07
============================================================



LTC-USD                              2018-07-12 21:40:41.202000
```

(continues on next page)

(continued from previous page)

```
============================================================
   Price: $75.68            Size: 0.19211000      Side:  sell
Best ask: $75.74        Best bid: $75.68        Spread: $0.06
============================================================



LTC-USD                           2018-07-12 21:41:09.452000
============================================================
   Price: $75.71            Size: 0.63097536      Side:   buy
Best ask: $75.71        Best bid: $75.68        Spread: $0.03
============================================================

^C
$
```

6.3 WebSocket Public API Reference

The following is an API reference of CoPrA generated from Python source code and docstrings.

Warning: This is a *complete* reference of the *public* API of CoPrA. User code and applications should only rely on the public API, since internal APIs can (and will) change without any guarantees. Anything *not* listed here is considered a private API.

6.3.1 Module `copra.websocket`

class copra.websocket.**Channel**(*name*, *product_ids*)

A WebSocket channel.

A Channel object encapsulates the Coinbase Pro WebSocket channel name *and* one or more Coinbase Pro product ids.

To read about Coinbase Pro channels and the data they return, visit: https://docs.gdax.com/#channels

Variables

- **name** (*str*) – The name of the WebSocket channel.
- **product_ids** (*set of str*) – Product ids for the channel.

__init__(*name*, *product_ids*)

Parameters

- **name** (*str*) – The name of the WebSocket channel. Possible values are heatbeat, ticker, level2, full, matches, or user
- **product_ids** (*str or list of str*) – A single product id (eg., 'BTC-USD') or list of product ids (eg., ['BTC-USD', 'ETH-EUR', 'LTC-BTC'])

Raises **ValueError** – If name not valid or product ids is empty.

class copra.websocket.**Client**(*loop*, *channels*, *feed_url='wss://ws-feed.pro.coinbase.com:443'*, *auth=False*, *key=''*, *secret=''*, *passphrase=''*, *auto_connect=True*, *auto_reconnect=True*, *name='WebSocket Client'*)

Asyncronous WebSocket client for Coinbase Pro.

__init__(*loop*, *channels*, *feed_url='wss://ws-feed.pro.coinbase.com:443'*, *auth=False*, *key=''*, *secret=''*, *passphrase=''*, *auto_connect=True*, *auto_reconnect=True*, *name='WebSocket Client'*)

Parameters

- **loop** (*asyncio loop*) – The asyncio loop that the client runs in.
- **channels** (*Channel or list of Channels*) – The channels to initially subscribe to.
- **feed_url** (*str*) – The url of the WebSocket server. The defualt is copra.WebSocket.FEED_URL (wss://ws-feed.gdax.com)
- **auth** (*bool*) – Whether or not the (entire) WebSocket session is authenticated. If True, you will need an API key from the Coinbase Pro website. The default is False.
- **key** (*str*) – The API key to use for authentication. Required if auth is True. The default is ''.
- **secret** (*str*) – The secret string for the API key used for authenticaiton. Required if auth is True. The default is ''.
- **passphrase** (*str*) – The passphrase for the API key used for authentication. Required if auth is True. The default is ''.
- **auto_connect** (*bool*) – If True, the Client will automatically add itself to its event loop (ie., open a connection if the loop is running or as soon as it starts). If False, add_as_task_to_loop() needs to be explicitly called to add the client to the loop. The default is True.
- **auto_reconnect** (*bool*) – If True, the Client will attemp to autom- matically reconnect and resubscribe if the connection is closed any way but by the Client explicitly itself. The default is True.
- **name** (*str*) – A name to identify this client in logging, etc.

Raises **ValueError** – If auth is True and key, secret, and passphrase are not provided.

add_as_task_to_loop()

Add the client to the asyncio loop.

Creates a coroutine for making a connection to the WebSocket server and adds it as a task to the asyncio loop.

close()

Close the WebSocket connection.

on_close(*was_clean*, *code*, *reason*)

Callback fired when the WebSocket connection has been closed.

(WebSocket closing handshake has been finished or the connection was closed uncleanly).

Parameters

- **was_clean** (*bool*) – True iff the WebSocket connection closed cleanly.
- **code** (*int or None*) – Close status code as sent by the WebSocket peer.
- **reason** (*str or None*) – Close reason as sent by the WebSocket peer.

on_error(*message*, *reason=''*)
Callback fired when an error message is received.

Parameters

- **message** (*str*) – A general description of the error.
- **reason** (*str*) – A more detailed description of the error.

on_message(*message*)
Callback fired when a complete WebSocket message was received.

You will likely want to override this method.

Parameters **message** (*dict*) – Dictionary representing the message.

on_open()
Callback fired on initial WebSocket opening handshake completion.

The WebSocket is open. This method sends the subscription message to the server.

subscribe(*channels*)
Subscribe to the given channels.

Parameters **channels** (*Channel or list of Channels*) – The channels to subscribe to.

unsubscribe(*channels*)
Unsubscribe from the given channels.

Parameters **channels** (*Channel or list of Channels*) – The channels to subscribe to.

CHAPTER 7

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

7.1 Types of Contributions

7.1.1 Report Bugs

Report bugs at https://github.com/tpodlaski/copra/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

7.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

7.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

7.1.4 Write Documentation

Coinbase Pro Asyncronous Websocket Client could always use more documentation, whether as part of the official Coinbase Pro Asyncronous Websocket Client docs, in docstrings, or even on the web in blog posts, articles, and such.

7.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/tpodlaski/copra/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

7.2 Get Started!

Ready to contribute? Here's how to set up *copra* for local development.

1. Fork the *copra* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/copra.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv copra
$ cd copra/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 copra tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

7.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7, 3.4, 3.5 and 3.6, and for PyPy. Check https://travis-ci.org/tpodlaski/copra/pull_requests and make sure that the tests pass for all supported Python versions.

7.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_copra
```

7.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER 8

Credits

8.1 Development Lead

- Tony Podlaski <tony@podlaski.com>

8.2 Contributors

None yet. Why not be the first?

CHAPTER 9

License

CHAPTER 10

History

10.1 0.1.0 (2018-07-06)

- First release on PyPI.

10.2 0.2.0 (2018-07-07)

- Added Client authentication.

10.3 0.3.0 (2018-07-09)

- Added reconnect option to Client.

10.4 0.4.0 (2018-07-10)

- Added subscribe and unsubscribe methods to Client.

10.5 1.0.0 (2018-07-12)

- Added full documentation of the CoPrA API.

10.6 1.0.1 (2018-07-12)

- Fixed typos in the documentation.

10.7 1.0.2 (2018-07-12)

- Added Examples page to the documentation.

10.8 1.0.3 (2018-07-16)

- More documentation typos fixed.

10.9 1.0.4 - 1.0.5 (2018-07-17)

- Non-API changes.

10.10 1.0.6 (2018-08-19)

- Updated Autobahn requirement to 18.8.1

10.11 1.0.7 (2018-08-19)

- Modified Travis config to test against Python 3.7.

10.12 1.1.0 (2018-11-27)

- Added REST client.

10.13 1.1.2 (2018-12-01)

- Updated documentation formatting.

10.14 1.2.0 (2019-01-04)

- Created copra.rest package and moved old copra.rest module to copra.rest.client.
- Created copra.websocket package and moved old copra.websocket module to copra.websocket.client.
- Add imports to copra.rest.__init__ and copra.websocket.__init__ so that classes and attributes can still be imported as they were before.
- Rewrote and completed unit tests from copra.websocket.

10.15 1.2.5 (2019-01-05)

- Updated copra.websocket.client unit tests to ignore those that are incompatible with Python 3.5 due to Mock methods that were not yet implemented.

10.16 1.2.6 (2019-01-07)

- Updated the REST client to attach an additional query string parameter to all GET requests. The parameter, 'no-cache', is a timestamp and ensures that the Coinbase server responds to all GET requests with fresh and not cached content.

Python Module Index

C

Index

Symbols

A

C

D

F

G

H

L

M

O

P

R

S

T

U

W