
coinbasepro-python

build unknown

The Python client for the Coinbase Pro API (formerly known as the GDAX)

Provided under MIT License by Daniel Paquin. *Note: this library may be subtly broken or buggy. The code is released under the MIT License – please take the following message to heart: > THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.*

Benefits

- A simple to use python wrapper for both public and authenticated endpoints.
- In about 10 minutes, you could be programmatically trading on one of the largest Bitcoin exchanges in the *world*!
- Do not worry about handling the nuances of the API with easy-to-use methods for every API endpoint.
- Gain an advantage in the market by getting under the hood of CB Pro to learn what and who is behind every tick.

Under Development

- Test Scripts
- Additional Functionality for the real-time order book
- FIX API Client **Looking for assistance**

Getting Started

This README is documentation on the syntax of the python client presented in this repository. See function docstrings for full syntax details.

This API attempts to present a clean interface to CB Pro, but in order to use it to its full potential, you must familiarize yourself with the official CB Pro documentation.

-
- <https://docs.pro.coinbase.com/>
 - You may manually install the project or use `pip`:

```
1 pip install cbpro
2 #or
3 pip install git+git://github.com/danpaquin/coinbasepro-python.git
```

Public Client

Only some endpoints in the API are available to everyone. The public endpoints can be reached using `PublicClient`

```
1 import cbpro
2 public_client = cbpro.PublicClient()
```

PublicClient Methods

- `get_products`

```
1 public_client.get_products()
```

- `get_product_order_book`

```
1 # Get the order book at the default level.
2 public_client.get_product_order_book('BTC-USD')
3 # Get the order book at a specific level.
4 public_client.get_product_order_book('BTC-USD', level=1)
```

- `get_product_ticker`

```
1 # Get the product ticker for a specific product.
2 public_client.get_product_ticker(product_id='ETH-USD')
```

- `get_product_trades (paginated)`

```
1 # Get the product trades for a specific product.
2 # Returns a generator
3 public_client.get_product_trades(product_id='ETH-USD')
```

- `get_product_historic_rates`

```
1 public_client.get_product_historic_rates('ETH-USD')
2 # To include other parameters, see function docstring:
3 public_client.get_product_historic_rates('ETH-USD', granularity=3000)
```

-
- `get_product_24hr_stats`

```
1 public_client.get_product_24hr_stats('ETH-USD')
```

- `get_currencies`

```
1 public_client.get_currencies()
```

- `get_time`

```
1 public_client.get_time()
```

Authenticated Client

Not all API endpoints are available to everyone. Those requiring user authentication can be reached using `AuthenticatedClient`. You must setup API access within your account settings. The `AuthenticatedClient` inherits all methods from the `PublicClient` class, so you will only need to initialize one if you are planning to integrate both into your script.

```
1 import cbpro
2 auth_client = cbpro.AuthenticatedClient(key, b64secret, passphrase)
3 # Use the sandbox API (requires a different set of API access
  credentials)
4 auth_client = cbpro.AuthenticatedClient(key, b64secret, passphrase,
5                                         api_url="https://api-public.sandbox.
                                              pro.coinbase.com")
```

Pagination

Some calls are paginated, meaning multiple calls must be made to receive the full set of data. The CB Pro Python API provides an abstraction for paginated endpoints in the form of generators which provide a clean interface for iteration but may make multiple HTTP requests behind the scenes. The pagination options `before`, `after`, and `limit` may be supplied as keyword arguments if desired, but aren't necessary for typical use cases.

```
1 fills_gen = auth_client.get_fills()
2 # Get all fills (will possibly make multiple HTTP requests)
3 all_fills = list(fills_gen)
```

One use case for pagination parameters worth pointing out is retrieving only new data since the previous request. For the case of `get_fills()`, the `trade_id` is the parameter used for indexing. By passing `before=some_trade_id`, only fills more recent than that `trade_id` will be returned.

Note that when using `before`, a maximum of 100 entries will be returned - this is a limitation of CB Pro.

```
1 from itertools import islice
2 # Get 5 most recent fills
3 recent_fills = islice(auth_client.get_fills(), 5)
4 # Only fetch new fills since last call by utilizing `before` parameter.
5 new_fills = auth_client.get_fills(before=recent_fills[0]['trade_id'])
```

AuthenticatedClient Methods

- `get_accounts`

```
1 auth_client.get_accounts()
```

- `get_account`

```
1 auth_client.get_account("7d0f7d8e-dd34-4d9c-a846-06f431c381ba")
```

- `get_account_history` (paginated)

```
1 # Returns generator:
2 auth_client.get_account_history("7d0f7d8e-dd34-4d9c-a846-06f431c381ba")
```

- `get_account_holds` (paginated)

```
1 # Returns generator:
2 auth_client.get_account_holds("7d0f7d8e-dd34-4d9c-a846-06f431c381ba")
```

- `buy & sell`

```
1 # Buy 0.01 BTC @ 100 USD
2 auth_client.buy(price='100.00', #USD
3                 size='0.01', #BTC
4                 order_type='limit',
5                 product_id='BTC-USD')
```

```
1 # Sell 0.01 BTC @ 200 USD
2 auth_client.sell(price='200.00', #USD
3                 size='0.01', #BTC
4                 order_type='limit',
5                 product_id='BTC-USD')
```

```
1 # Limit order-specific method
2 auth_client.place_limit_order(product_id='BTC-USD',
3                               side='buy',
```

```
4         price='200.00',
5         size='0.01')
```

```
1 # Place a market order by specifying amount of USD to use.
2 # Alternatively, `size` could be used to specify quantity in BTC amount
3
4 auth_client.place_market_order(product_id='BTC-USD',
5                                side='buy',
6                                funds='100.00')
```

```
1 # Stop order. `funds` can be used instead of `size` here.
2 auth_client.place_stop_order(product_id='BTC-USD',
3                               stop_type='loss',
4                               price='200.00',
5                               size='0.01')
```

- `cancel_order`

```
1 auth_client.cancel_order("d50ec984-77a8-460a-b958-66f114b0de9b")
```

- `cancel_all`

```
1 auth_client.cancel_all(product_id='BTC-USD')
```

- `get_orders` (paginated)

```
1 # Returns generator:
2 auth_client.get_orders()
```

- `get_order`

```
1 auth_client.get_order("d50ec984-77a8-460a-b958-66f114b0de9b")
```

- `get_fills` (paginated)

```
1 # All return generators
2 auth_client.get_fills()
3 # Get fills for a specific order
4 auth_client.get_fills(order_id="d50ec984-77a8-460a-b958-66f114b0de9b")
5 # Get fills for a specific product
6 auth_client.get_fills(product_id="ETH-BTC")
```

- `deposit & withdraw`

```
1 depositParams = {
2     'amount': '25.00', # Currency determined by account specified
3     'coinbase_account_id': '60680c98bfe96c2601f27e9c'
```

```
4 }
5 auth_client.deposit(depositParams)
```

```
1 # Withdraw from CB Pro into Coinbase Wallet
2 withdrawParams = {
3     'amount': '1.00', # Currency determined by account specified
4     'coinbase_account_id': '536a541fa9393bb3c7000023'
5 }
6 auth_client.withdraw(withdrawParams)
```

WebsocketClient

If you would like to receive real-time market updates, you must subscribe to the websocket feed.

Subscribe to a single product

```
1 import cbpro
2
3 # Parameters are optional
4 wsClient = cbpro.WebsocketClient(url="wss://ws-feed.pro.coinbase.com",
5                                   products="BTC-USD",
6                                   channels=["ticker"])
7 # Do other stuff...
8 wsClient.close()
```

Subscribe to multiple products

```
1 import cbpro
2 # Parameters are optional
3 wsClient = cbpro.WebsocketClient(url="wss://ws-feed.pro.coinbase.com",
4                                   products=["BTC-USD", "ETH-USD"],
5                                   channels=["ticker"])
6 # Do other stuff...
7 wsClient.close()
```

WebsocketClient + Mongoddb

The `WebsocketClient` now supports data gathering via MongoDB. Given a MongoDB collection, the `WebsocketClient` will stream results directly into the database collection.

```
1 # import PyMongo and connect to a local, running Mongo instance
2 from pymongo import MongoClient
3 import cbpro
4 mongo_client = MongoClient('mongodb://localhost:27017/')
5
6 # specify the database and collection
7 db = mongo_client.cryptocurrency_database
```

```
8 BTC_collection = db.BTC_collection
9
10 # instantiate a WebSocketClient instance, with a Mongo collection as a
    parameter
11 wsClient = cbpro.WebsocketClient(url="wss://ws-feed.pro.coinbase.com",
    products="BTC-USD",
12     mongo_collection=BTC_collection, should_print=False)
13 wsClient.start()
```

WebSocketClient Methods

The `WebSocketClient` subscribes in a separate thread upon initialization. There are three methods which you could overwrite (before initialization) so it can react to the data streaming in. The current client is a template used for illustration purposes only.

- `onOpen` - called once, *immediately before* the socket connection is made, this is where you want to add initial parameters.
- `onMessage` - called once for every message that arrives and accepts one argument that contains the message of dict type.
- `on_close` - called once after the websocket has been closed.
- `close` - call this method to close the websocket connection (do not overwrite).

```
1 import cbpro, time
2 class myWebSocketClient(cbpro.WebsocketClient):
3     def on_open(self):
4         self.url = "wss://ws-feed.pro.coinbase.com/"
5         self.products = ["LTC-USD"]
6         self.message_count = 0
7         print("Lets count the messages!")
8     def on_message(self, msg):
9         self.message_count += 1
10        if 'price' in msg and 'type' in msg:
11            print ("Message type:", msg["type"],
12                "\t@ {:.3f}".format(float(msg["price"])))
13    def on_close(self):
14        print("-- Goodbye! --")
15
16 wsClient = myWebSocketClient()
17 wsClient.start()
18 print(wsClient.url, wsClient.products)
19 while (wsClient.message_count < 500):
20     print ("\nmessage_count =", "{} \n".format(wsClient.message_count))
21     time.sleep(1)
22 wsClient.close()
```

Testing

A test suite is under development. Tests for the authenticated client require a set of sandbox API credentials. To provide them, rename `api_config.json.example` in the tests folder to `api_config.json` and edit the file accordingly. To run the tests, start in the project directory and run

```
1 python -m pytest
```

Real-time OrderBook

The `OrderBook` is a convenient data structure to keep a real-time record of the orderbook for the `product_id` input. It processes incoming messages from an already existing `WebsocketClient`. Please provide your feedback for future improvements.

```
1 import cbpro, time, Queue
2 class myWebsocketClient(cbpro.WebsocketClient):
3     def on_open(self):
4         self.products = ['BTC-USD', 'ETH-USD']
5         self.order_book_btc = OrderBookConsole(product_id='BTC-USD')
6         self.order_book_eth = OrderBookConsole(product_id='ETH-USD')
7     def on_message(self, msg):
8         self.order_book_btc.process_message(msg)
9         self.order_book_eth.process_message(msg)
10
11 wsClient = myWebsocketClient()
12 wsClient.start()
13 time.sleep(10)
14 while True:
15     print(wsClient.order_book_btc.get_ask())
16     print(wsClient.order_book_eth.get_bid())
17     time.sleep(1)
```

Testing

Unit tests are under development using the pytest framework. Contributions are welcome!

To run the full test suite, in the project directory run:

```
1 python -m pytest
```

Change Log

1.1.2 Current PyPI release - Refactor project for Coinbase Pro - Major overhaul on how pagination is handled

1.0 - The first release that is not backwards compatible - Refactored to follow PEP 8 Standards - Improved Documentation

0.3 - Added crypto and LTC deposit & withdraw (undocumented). - Added support for Margin trading (undocumented). - Enhanced functionality of the WebsocketClient. - Soft launch of the OrderBook (undocumented). - Minor bug squashing & syntax improvements.

0.2.2 - Added additional API functionality such as cancelAll() and ETH withdrawal.

0.2.1 - Allowed `WebsocketClient` to operate intuitively and restructured example workflow.

0.2.0 - Renamed project to GDAX-Python - Merged Websocket updates to handle errors and reconnect.

0.1.2 - Updated JSON handling for increased compatibility among some users. - Added support for payment methods, reports, and Coinbase user accounts. - Other compatibility updates.

0.1.1b2 - Original PyPI Release.