

The unofficial Python client for the GDAX API

[#gdax](#) [#python-client](#) [#websocket-client](#) [#gdax-api](#) [#coinbase](#) [#exchange](#) [#wrapper](#) [#libaray](#) [#orderbook](#) [#bitcoin](#) [#ethereum](#) [#trading](#)

🕒 165 commits

🌿 1 branch

📦 0 releases

👤 19 contributors

📄 MIT

Branch: master ▾












New pull request

Create new file

Upload files

Find file

Clone or download ▾

	danpaquin committed on GitHub Merge pull request #95 from bianjiang/master ...	Latest commit 05ce2e2 on Jul 31
 gdax	Merge pull request #95 from bianjiang/master	2 months ago
 tests	Clean up PublicClient. Add docstrings + tests.	3 months ago
 .gitignore	Added authentication functionality for websocket	2 months ago
 LICENSE	Documentation Updates	4 months ago
 README.md	Added authentication functionality for websocket	2 months ago
 __init__.py	Added get_current_book, Fixed issue with segfault, Fixed issue with N...	4 months ago
 contributors.txt	Hotfix: KeyError on 'remaining_size'	4 months ago
 requirements.txt	Intial commit for real-time order book	4 months ago
 setup.cfg	Added project to PyPI	a year ago
 setup.py	Updated files and refactored folder names for v1.0.6	3 months ago

README.md

gdax-python

The Python client for the [GDAX API](#) (formerly known as the Coinbase Exchange API)

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 GDAX to learn what and who is *really* 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 GDAX, but in order to use it to its full potential, you must familiarize yourself with the official GDAX documentation.

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

```
pip install gdax
```

Public Client

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

```
import gdax
public_client = gdax.PublicClient()
```

PublicClient Methods

- [get_products](#)

```
public_client.get_products()
```

- [get_product_order_book](#)

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

- [get_product_ticker](#)

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

- [get_product_trades](#)

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

- [get_product_historic_rates](#)

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

- [get_product_24hr_stats](#)

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

- [get_currencies](#)

```
public_client.get_currencies()
```

- [get_time](#)

```
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.

```
import gdax
auth_client = gdax.AuthenticatedClient(key, b64secret, passphrase)
# Set a default product
auth_client = gdax.AuthenticatedClient(key, b64secret, passphrase,
                                       product_id="ETH-USD")
# Use the sandbox API (requires a different set of API access credentials)
auth_client = gdax.AuthenticatedClient(key, b64secret, passphrase,
                                       api_url="https://api-public.sandbox.gdax.com")
```

Pagination

Some calls are [paginated](#), meaning multiple calls must be made to receive the full set of data. Each page/request is a list of dict objects that are then appended to a master list, making it easy to navigate pages (e.g. `request[0]` would return the first page of data in the example below). *This feature is under consideration for redesign. Please provide feedback if you have issues or suggestions*

```
request = auth_client.get_fills(limit=100)
request[0] # Page 1 always present
request[1] # Page 2+ present only if the data exists
```

It should be noted that limit does not behave exactly as the official documentation specifies. If you request a limit and that limit is met, additional pages will not be returned. This is to ensure speedy response times when less data is preferred.

AuthenticatedClient Methods

- [get_accounts](#)

```
auth_client.get_accounts()
```

- [get_account](#)

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

- [get_account_history](#) (paginated)

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

- [get_account_holds](#) (paginated)

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

- [buy & sell](#)

```
# Buy 0.01 BTC @ 100 USD
auth_client.buy(price='100.00', #USD
               size='0.01', #BTC
               product_id='BTC-USD')
```

```
# Sell 0.01 BTC @ 200 USD
auth_client.sell(price='200.00', #USD
                size='0.01', #BTC
                product_id='BTC-USD')
```

- [cancel_order](#)

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

- [cancel_all](#)

```
auth_client.cancel_all(product='BTC-USD')
```

- [get_orders](#) (paginated)

```
auth_client.get_orders()
```

- [get_order](#)

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

- [get_fills](#) (paginated)

```
auth_client.get_fills()
# Get fills for a specific order
auth_client.get_fills(order_id="d50ec984-77a8-460a-b958-66f114b0de9b")
# Get fills for a specific product
auth_client.get_fills(product_id="ETH-BTC")
```

- [deposit & withdraw](#)

```
gdax
depositParams = {
    'amount': '25.00', # Currency determined by account specified
    'coinbase_account_id': '60680c98bfe96c2601f27e9c'
}
auth_client.deposit(depositParams)
```

```
# Withdraw from GDAX into Coinbase Wallet
withdrawParams = {
    'amount': '1.00', # Currency determined by account specified
    'coinbase_account_id': '536a541fa9393bb3c7000023'
}
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

```
import gdax
# Parameters are optional
wsClient = gdax.WebsocketClient(url="wss://ws-feed.gdax.com", products="BTC-USD")
# Do other stuff...
wsClient.close()
```

Subscribe to multiple products

```
import gdax
# Parameters are optional
wsClient = gdax.WebsocketClient(url="wss://ws-feed.gdax.com",
                                products=["BTC-USD", "ETH-USD"])
# Do other stuff...
wsClient.close()
```

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.
- `onClose` - called once after the websocket has been closed.
- `close` - call this method to close the websocket connection (do not overwrite).

```
import gdax, time
class myWebsocketClient(gdax.WebsocketClient):
    def on_open(self):
        self.url = "wss://ws-feed.gdax.com/"
        self.products = ["LTC-USD"]
        self.message_count = 0
        print("Lets count the messages!")
    def on_message(self, msg):
        self.message_count += 1
        if 'price' in msg and 'type' in msg:
            print ("Message type:", msg["type"],
                  "\t@ {}".format(float(msg["price"])))
    def on_close(self):
        print("-- Goodbye! --")

wsClient = myWebsocketClient()
wsClient.start()
print(wsClient.url, wsClient.products)
while (wsClient.message_count < 500):
    print ("\nmessage_count =", "{} \n".format(wsClient.message_count))
    time.sleep(1)
wsClient.close()
```

Testing

A test suite is under development. To run the tests, start in the project directory and run

```
python -m pytest
```

Real-time OrderBook

The `orderBook` subscribes to a websocket and keeps a real-time record of the orderbook for the `product_id` input. Please provide your feedback for future improvements.

```
import gdax, time
order_book = gdax.OrderBook(product_id='BTC-USD')
order_book.start()
time.sleep(10)
order_book.close()
```

Change Log

1.0 Current PyPI release

- 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.