quadriga Documentation

Release 2.1.0

Joohwan Oh

May 12, 2018

Contents

Welcome to the documentation for **quadriga**, Python client for Canadian cryptocurrency exchange platform QuadrigaCX. It wraps the exchange's REST API v2 using the requests library.

CHAPTER 1

Requirements

- Python 2.7, 3.4, 3.5 or 3.6
- QuadrigaCX API secret, API key and client ID (the number used for your login)

CHAPTER 2

Installation

To install a stable version from PyPi:

```
~$ pip install quadriga
```

To install the latest version directly from GitHub:

```
~$ pip install -e git+git@github.com:joowani/quadriga.git@master#egg=quadriga
```

You may need to use `sudo` depending on your environment.

CHAPTER 3

Contents

3.1 Getting Started

Here is an example showing how a **quadriga** client can be initialized and used:

```
from quadriga import QuadrigaClient

client = QuadrigaClient(
    api_key='api_key',
    api_secret='api_secret',
    client_id='client_id',
)
client.get_balance()                # Get the user's account balance
client.lookup_order(['order_id'])   # Look up one or more orders by ID
client.cancel_order('order_id')     # Cancel an order by ID

client.get_deposit_address('bch')   # Get the funding address for BCH
client.get_deposit_address('btc')   # Get the funding address for BTC
client.get_deposit_address('btg')   # Get the funding address for BTG
client.get_deposit_address('eth')   # Get the funding address for ETH
client.get_deposit_address('ltc')   # Get the funding address for LTC

client.withdraw('bch', 1, 'bch_wallet_address')  # Withdraw 1 BCH to wallet
client.withdraw('btc', 1, 'btc_wallet_address')  # Withdraw 1 BTC to wallet
client.withdraw('btg', 1, 'btg_wallet_address')  # Withdraw 1 BTG to wallet
client.withdraw('eth', 1, 'eth_wallet_address')  # Withdraw 1 ETH to wallet
client.withdraw('ltc', 1, 'ltc_wallet_address')  # Withdraw 1 LTC to wallet

book = client.book('btc_cad')
book.get_ticker()                   # Get the latest ticker information
book.get_user_orders()              # Get user's open orders
book.get_user_trades()              # Get user's transactions
book.get_public_orders()            # Get public open orders
book.get_public_trades()            # Get recent public transactions
```

(continues on next page)

(continued from previous page)

```
book.buy_market_order(10)           # Buy 10 BTC at market price
book.buy_limit_order(5, 10)         # Buy 5 BTC at limit price of $10 CAD
book.sell_market_order(10)          # Sell 10 BTC at market price
book.sell_limit_order(5, 10)        # Sell 5 BTC at limit price of $10 CAD
```

See *API Specification* for more details.

3.2 API Specification

3.2.1 QuadrigaClient

class quadriga.client.**QuadrigaClient**(*api_key=None*, *api_secret=None*, *client_id=None*, *timeout=None*, *session=None*, *logger=None*)

Python client for QuadrigaCX's REST API v2.

Parameters

- **api_key** (*str | unicode*) – QuadrigaCX API key.
- **api_secret** (*str | unicode*) – QuadrigaCX API secret.
- **client_id** (*str | unicode | int*) – QuadrigaCX client ID (number used for user login).
- **timeout** (*int | float*) – Number of seconds to wait for QuadrigaCX to respond to an API request.
- **session** (*requests.Session*) – User-defined requests.Session object. If not set, `requests.Session()` is used by default.
- **logger** (*logging.Logger*) – Logger to record debug messages with. If not set, `logging.getLogger('quadriga')` is used by default.

Variables

- **version** (*str | unicode*) – Client version.
- **url** (*str | unicode*) – QuadrigaCX API base URL.
- **order_books** (*[str | unicode]*) – Order books supported.
- **major_currencies** (*dict*) – Major currencies supported.

Note: Parameters **api_key**, **api_secret** and **client_id** are optional. See *Public API* for details.

---

book(*name*)

Return an API wrapper for the given order book.

Parameters **name** (*str | unicode*) – Order book name (e.g. "btc_cad").

Returns Order book API wrapper.

Return type *quadriga.book.OrderBook*

Raises ***InvalidOrderBookError*** – If an invalid order book is given.

Example:

```
>>> from quadriga import QuadrigaClient
>>>
>>> client = QuadrigaClient()
>>>
>>> eth = client.book('eth_cad').get_ticker()
>>> btc = client.book('btc_cad').get_ticker()
```

cancel_order(*order_id*)
Cancel an open order by ID (64 hexadecmial characters).

Parameters **order_id** (*str | unicode*) – Order ID.

Returns True if the order was cancelled successfully.

Return type bool

get_balance()
Return user's account balance.

Returns User's account balance.

Return type dict

get_deposit_address(*currency*)
Return the deposit address for the given major currency.

Parameters **currency** (*str | unicode*) – Major currency name in lowercase (e.g. "btc", "eth").

Returns Deposit address.

Return type str | unicode

lookup_order(*order_id*)
Look up one or more orders by ID (64 hexadecmial characters).

Parameters **order_id** (*str | unicode | [str | unicode]*) – Order ID or list of order IDs.

Returns Order details.

Return type [dict]

withdraw(*currency*, *amount*, *address*)
Withdraw a major currency from QuadrigaCX to the given wallet.

Parameters

- **currency** (*str | unicode*) – Major currency name in lowercase (e.g. "btc", "eth").
- **amount** (*int | float | str | unicode | decimal.Decimal*) – Withdrawal amount.
- **address** (*str | unicode*) – Wallet address.

Warning: Specifying incorrect major currency or wallet address could result in permanent loss of your coins. Please be careful when using this method!

3.2.2 OrderBook

class quadriga.book.**OrderBook**(*name*, *rest_client*, *logger*)
API wrapper for an order book on QuadrigaCX.

This class is not meant to be imported or instantiated directly. Use method *quadriga.client.QuadrigaClient.book()* instead.

buy_limit_order(*amount*, *price*)
Place a buy order at the given limit price.

Parameters

- **amount** (*int | float | str | unicode | decimal.Decimal*) – Amount of major currency to buy at limit price.
- **price** (*int | float | str | unicode | decimal.Decimal*) – Limit price.

Returns Order details.

Return type dict

buy_market_order(*amount*)
Place a buy order at market price.

Parameters **amount** (*int | float | str | unicode | decimal.Decimal*) – Amount of major currency to buy at market price.

Returns Order details.

Return type dict

get_public_orders(*group=False*)
Return public orders that are currently open.

Parameters **group** (*bool*) – If set to True (default: False), orders with the same price are grouped.

Returns Public orders currently open.

Return type dict

get_public_trades(*time_frame=u'hour'*)
Return public trades that were completed recently.

Parameters **time_frame** (*str | unicode*) – Time frame. Allowed values are "minute" for trades in the last minute, or "hour" for trades in the last hour (default: "hour").

Returns Public trades completed recently.

Return type [dict]

get_ticker()
Return the latest ticker information.

Returns Latest ticker information.

Return type dict

get_user_orders()
Return user's orders that are currently open.

Returns User's orders currently open.

Return type [dict]

get_user_trades (*limit=0*, *offset=0*, *sort=u'desc'*)
Return user's trade history.

Parameters

- **limit** (*int*) – Maximum number of trades to return. If set to 0 or lower, all trades are returned (default: 0).
- **offset** (*int*) – Number of trades to skip.
- **sort** (*str | unicode*) – Method used to sort the results by date and time. Allowed values are "desc" for descending order, and "asc" for ascending order (default: "desc").

Returns User's trade history.

Return type [dict]

sell_limit_order (*amount*, *price*)
Place a sell order at the given limit price.

Parameters

- **amount** (*int | float | str | unicode | decimal.Decimal*) – Amount of major currency to sell at limit price.
- **price** (*int | float | str | unicode | decimal.Decimal*) – Limit price.

Returns Order details.

Return type dict

sell_market_order (*amount*)
Place a sell order at market price.

Parameters **amount** (*int | float | str | unicode | decimal.Decimal*) – Amount of major currency to sell at market price.

Returns Order details.

Return type dict

3.3 Error Handling

When an API request fails, *quadriga.exceptions.RequestError* is raised. The exception object contains the error message, error code and HTTP request response details.

Example:

```
from quadriga import QuadrigaClient
from quadriga.exceptions import RequestError

client = QuadrigaClient(
    api_key='api_key',
    api_secret='api_secret',
    client_id='client_id',
)
try:
    client.get_balance()    # Fails due to invalid credentials
except RequestError as err:
    err.message             # Error message
```

(continues on next page)

(continued from previous page)

```
    err.url                 # API endpoint
    err.body                # Raw response body from QuadrigaCX
    err.headers             # Response headers
    err.http_code           # HTTP status code
    err.error_code          # Error code from QuadrigaCX
    err.response            # requests.Response object
    err.response.request    # requests.Request object
```

3.3.1 Exceptions

Below are exceptions raised by quadriga client.

exception quadriga.exceptions.**InvalidCurrencyError**
Raised when an invalid major currency is given.

exception quadriga.exceptions.**InvalidOrderBookError**
Raised when an invalid order book is given.

exception quadriga.exceptions.**QuadrigaError**
Base (catch-all) client exception.

exception quadriga.exceptions.**RequestError**(*response*, *message*, *error_code=None*)
Raised when an API request to QuadrigaCX fails.

Variables

- **message** (*str | unicode*) – Error message.
- **url** (*str | unicode*) – QuadrigaCX API endpoint.
- **body** (*str | unicode*) – Raw response body from QuadrigaCX.
- **headers** (*requests.structures.CaseInsensitiveDict*) – Response headers.
- **http_code** (*int*) – HTTP status code.
- **error_code** (*int*) – Error code from QuadrigaCX.
- **response** (*requests.Response*) – Response object.

3.4 Public API

If only public API is required, you do not need to provide parameters **api_key**, **api_secret** and **client_id** when initializing *quadriga.client.QuadrigaClient*.

Example:

```
from quadriga import QuadrigaClient
from quadriga.exceptions import RequestError

# Initialize the client without credentials.
client = QuadrigaClient()

# Private (user) API calls fail.
try:
    client.book('btc_cad').get_user_orders()
```

(continues on next page)

(continued from previous page)

```
except RequestError as err:
    assert err.error_code == 101
try:
    client.book('btc_cad').get_user_trades()
except RequestError as err:
    assert err.error_code == 101

# Public API calls still go through.
orders = client.book('btc_cad').get_public_orders()
trades = client.book('btc_cad').get_public_trades()
```

3.5 Logging

By default, quadriga client logs API call history using the `quadriga` logger at `logging.DEBUG` level. You can customize this logger to suit your usecase.

Example:

```
import logging

from quadriga import QuadrigaClient

logger = logging.getLogger('quadriga')

# Set the logging level.
logger.setLevel(logging.DEBUG)

# Attach a handler.
handler = logging.StreamHandler()
formatter = logging.Formatter('[%(asctime)s] %(message)s')
handler.setFormatter(formatter)
logger.addHandler(handler)

# Inject the logger during client initialization.
client = QuadrigaClient(logger=logger)

client.book('eth_cad').get_ticker()
client.book('eth_cad').get_public_orders()
```

The logging output from above should look something like this:

```
[2017-04-12 23:55:52,230] eth_cad: get ticker
[2017-04-12 23:55:53,741] eth_cad: get public orders
```

To see full request details, turn on the logging for requests library.

Example:

```
import requests
import logging

try:
    # For Python 3.
    from http.client import HTTPConnection
except ImportError:
```

(continues on next page)

(continued from previous page)

```
    # For Python 2.
    from httplib import HTTPConnection
HTTPConnection.debuglevel = 1

logging.basicConfig()
logging.getLogger().setLevel(logging.DEBUG)
requests_log = logging.getLogger("requests.packages.urllib3")
requests_log.setLevel(logging.DEBUG)
requests_log.propagate = True
```

3.6 HTTP Session

You can use your own `requests.Session` objects for sending HTTP requests to QuadrigaCX. For example, let's say you want:

- Automatic retries
- Additional request header called `x-my-header`

You can initialize and inject your session as follows:

```
from requests.adapters import HTTPAdapter
from requests import Session

from quadriga import QuadrigaClient

session = Session()

# Enable automatic retries.
adapter = HTTPAdapter(max_retries=5)
session.mount('https://', adapter)

# Add your request header.
session.headers.update({'x-my-header': 'true'})

# Inject the session during client initialization.
client = QuadrigaClient(session=session)

client.book('eth_cad').get_ticker()
client.book('eth_cad').get_public_orders()
```

For more information on how to configure a `requests.Session` object, refer to requests documentation.

3.7 Contributing

3.7.1 Instructions

Before submitting a pull request on GitHub, please make sure you meet the following **requirements**:

- The pull request points to the dev (development) branch.
- All changes are squashed into a single commit (I like to use `git rebase -i` to do this).
- The commit message is in present tense (good: "Add feature", bad: "Added feature").

- Correct and consistent style: Sphinx-compatible docstrings, correct snake and camel casing, and PEP8 compliance (see below).
- No classes/methods/functions with missing docstrings or commented-out lines. You can take a look at the source code on GitHub for examples.
- The test coverage remains at %100. You may find yourself having to write superfluous unit tests to keep this number up. If a piece of code is trivial and has no need for tests, use this to exclude it from coverage.
- No build failures on TravisCI. The builds automatically trigger on PR submissions.
- Does not break backward-compatibility (unless there is a really good reason).
- Compatibility with all supported Python versions: 2.7, 3.4, 3.5 and 3.6.

Warning: The dev branch is occasionally rebased, and its commit history may be overwritten in the process. Before you begin feature work, git fetch or pull to ensure that your local branch has not diverged. If you see git conflicts and just want to start from scratch, run these commands:

```
~$ git checkout dev
~$ git fetch origin
~$ git reset --hard origin/dev  # THIS WILL WIPE ALL LOCAL CHANGES
```

3.7.2 Style

To ensure PEP8 compliance, run flake8:

```
~$ pip install flake8
~$ git clone https://github.com/joowani/quadriga.git
~$ cd quadriga
~$ flake8
```

You must resolve all issues reported. If there is a good reason to ignore errors coming from a specific piece of code, visit here to see how to exclude the lines.

3.7.3 Testing

To test your changes, run the unit tests that come with **quadriga** on your local machine. The tests use pytest.

To run the unit tests:

```
~$ pip install pytest
~$ git clone https://github.com/joowani/quadriga.git
~$ cd quadriga
~$ py.test --verbose
```

To run the unit tests with coverage report:

```
~$ pip install coverage pytest pytest-cov
~$ git clone https://github.com/joowani/quadriga.git
~$ cd quadriga
~$ py.test --verbose --cov=quadriga --cov-report=html

# Open the generated file htmlcov/index.html in a browser
```

3.7.4 Documentation

The documentation (including the README) is written in reStructuredText and uses Sphinx. To build an HTML version of the documentation on your local machine:

```
~$ pip install sphinx sphinx_rtd_theme
~$ git clone https://github.com/joowani/quadriga.git
~$ cd quadriga/docs
~$ sphinx-build . build

# Open the generated file build/index.html in a browser
```

As always, thanks for your contribution!

CHAPTER 4

Disclaimer

The author(s) of this project is in no way affiliated with QuadrigaCX, and shall not accept any liability, obligation or responsibility whatsoever for any cost, loss or damage arising from the use of this client. Please use at your own risk.

Python Module Index

q

Index