trading-ig

Sep 01, 2023

Contents

A lightweight Python wrapper for the IG Markets API. Simplifies access to the IG REST and Streaming APIs.

CHAPTER 1

What is it?

IG Markets provides financial spread betting and CFD platforms for trading equities, forex, commodities, indices, cryptocurrencies, bonds, rates, options and more.

IG provide APIs so that developers can access their platforms programmatically. Using the APIs you can get live and historical data, automate your trades, or create apps. For details about the IG APIs please see their site:

https://labs.ig.com/

NOTE: this is not an IG project. Use it at your own risk

CHAPTER 2

Dependencies

A number of dependencies in this project are marked as 'optional', this is *by design*. There is a brief explanation in *this FAQ item*.

For full details, see pyproject.toml

CHAPTER 3

Support

This is an open source project, maintained by busy volunteers in their spare time. It should make it easier and quicker for you to use the IG APIs. However, it is not a support resource for users of the IG APIs. IG have their own support site, and an API support team.

If you have a problem when using this library, please think carefully about the likely cause. If it is:

- a bug in this library
- a request for a feature
- missing or confusing documentation

then please create an issue. See the guidance notes *here*.

For anything else, please use the documentation and tools provided by IG

- Getting Started Guide
- REST API Guide
- REST API Reference
- REST API Companion
- Streaming API Guide
- Streaming API Reference
- Streaming API Companion

CHAPTER 4

Contributing

Contributions are welcome. We use automated formatting, linting and testing so please make your code passes the tests, see *here*. And ideally, write some tests for the new code or feature.

CHAPTER 5

License

BSD (See LICENSE)

CHAPTER 6

See also

6.1 Quickstart

6.1.1 Installation

This project uses Poetry.

Adding to an existing Poetry project:

```
$ poetry add trading-ig
```

With all the optional dependencies:

```
$ poetry add trading-ig[pandas,munch,tenacity]
```

Cloning the project with Poetry:

```
$ git clone https://github.com/ig-python/trading-ig
$ cd trading-ig
$ poetry install
```

And with all optional dependencies:

```
$ poetry install --extras "pandas munch tenacity"
```

Installing with pip:

```
$ pip install trading-ig
```

And with all optional dependencies:

```
$ pip install trading-ig pandas munch tenacity
```

6.1.2 Configuration

Config file

Make a copy of the config template module (`trading_ig_config.default.py`), and rename `trading_ig_config.py`. Edit the new file, replacing the variables with your own:

```
class config(object):
    username = "your_username"
    password = "your_password"
    api_key = "your_api_key"
    acc_type = "DEMO"
    acc_number = "your_account_number"
```

Environment variables

If any exceptions are raised loading the config, IGService will attempt to find your authentication details from environment variables

```
$ export IG_SERVICE_USERNAME="your_username"
$ export IG_SERVICE_PASSWORD="your_password"
$ export IG_SERVICE_API_KEY="your_api_key"
$ export IG_SERVICE_ACC_NUMBER="your_account_number"
```

6.1.3 Connection

```
>>> from trading_ig.rest import IGService
>>> from trading_ig.config import config
>>> ig_service = IGService(config.username, config.password, config.api_key)
>>> ig = ig_service.create_session()
>>> ig
```

6.1.4 Using the REST API

Searching for a market

```
>>> results = ig_service.search_markets("gold")
>>> results
```

Get info about a market

```
>>> market = ig_service.fetch_market_by_epic('CS.D.USCGC.TODAY.IP')
>>> market
```

Getting historic prices

```
>>> result = ig_service.fetch_historical_prices_by_epic(epic='CS.D.USCGC.TODAY.IP')
>>> result['prices']
```

Opening a position

```
>>> resp = ig_service.create_open_position(
        currency_code='GBP',
        direction='BUY',
        epic='CS.D.USCGC.TODAY.IP',
        order_type='MARKET',
        expiry='DFB',
        force_open='false',
        guaranteed_stop='false',
        size=0.5, level=None,
        limit_distance=None,
        limit_level=None,
        quote_id=None,
        stop_level=None,
        stop_distance=None,
        trailing_stop=None,
        trailing_stop_increment=None)
>>> resp
```

Getting account activity

```
>>> from_date = datetime(2021, 1, 1)
>>> activities = ig_service.fetch_account_activity(from_date=from_date)
>>> activities
```

6.1.5 Using the Streaming API

Assuming config as above

```
>>> from trading_ig import IGService, IGStreamService
>>> from trading_ig.config import config
>>> from trading_ig.lightstreamer import Subscription
```

```
>>> def on_update(item):
>>>     print("{UPDATE_TIME:<8} {stock_name:<19} Bid {BID:>5} Ask {OFFER:>5}".
↪format(stock_name=item["name"], **item["values"]))
```

```
>>> ig_service = IGService(config.username, config.password, config.api_key, config.
↪acc_type, acc_number=config.acc_number)
>>> ig_stream_service = IGStreamService(ig_service)
>>> ig_stream_service.create_session()
>>> sub = Subscription(mode="MERGE", items=["L1:CS.D.GBPUSD.TODAY.IP"], fields=[
↪"UPDATE_TIME", "BID", "OFFER"])
>>> sub.addlistener(on_update)
>>> ig_stream_service.ls_client.subscribe(sub)
>>> ig_stream_service.disconnect()
```

6.2 FAQs

6.2.1 Why are there multiple versions of some API endpoints?

IG have updated their API endpoints at various times, and they do not close down the older ones - presumably to not break anyone's applications. They have been updated for various reasons, including

- to support UTC time stamps
- to offer more options
- to include paged sets of results
- to incorporate multiple features into one endpoint

Generally a higher version number is better - the exception is session creation, see *here*

6.2.2 Are there any code samples?

There are a few simple ones in the *Quickstart*. There are also a few in the `/sample` directory, see *here*. And there are a lot of usage samples in the unit and integration tests.

There are not many samples using the Streaming API. If you use the Streaming API, please consider contributing.

6.2.3 How do I run the code samples?

For the REST API sample - from the project root, run:

```
$ python sample/rest_ig.py
```

For the streaming sample, run:

```
$ python sample/stream_ig.py
```

In both samples there are commented lines with alternative CFD epics

6.2.4 How to see log messages from `ig_trading`?

In your code, do something like:

```
import logging

logging.basicConfig(
        level=logging.INFO,
        format='%(asctime)s %(levelname)s %(message)s',
        datefmt='%Y-%m-%d %H:%M:%S')

logging.info("Log something")
```

6.2.5 Why did I get a `KeyError: CST`?

`CST` is the name of one of the HTTP headers returned in the response from IG on successful connection. If you see this error it probably means your username, password or API key is wrong. Or that you are using the DEMO credentials to connect to the LIVE account, or vice versa.

6.2.6 Why do I get a error like `error.public-api.exceeded-*-allowance`?

The IG APIs have rate limits; you can only make a certain number of requests during a certain time period (eg minute, hour, week etc) depending on the request type. The four error types are:

- `error.public-api.exceeded-api-key-allowance`

- error.public-api.exceeded-account-allowance
- error.public-api.exceeded-account-trading-allowance
- error.public-api.exceeded-account-historical-data-allowance

The limits for the LIVE environment are published here, but the limits for DEMO are lower, and have been known to change randomly and without notice. If you see one of these errors, you have exceeded one of the limits.

You can query the limits associated with either a LIVE or DEMO API key after logging on by calling:

IGService.get_client_apps()

This is the rate used when a new IGService object is created with the use_rate_limiter kwarg set as True.

6.2.7 How to avoid hitting the rate limits?

There are three options.

The first is manage it yourself with your own code.

Secondly, since version 0.0.10, `trading-ig` has support for tenacity, a general purpose retrying library. You can initialise the `IGService` class with a `Retrying` instance, like:

```python
from trading_ig.rest import IGService, ApiExceededException
from tenacity import Retrying, wait_exponential, retry_if_exception_type

retryer = Retrying(wait=wait_exponential(),
    retry=retry_if_exception_type(ApiExceededException))
ig_service = IGService('username', 'password', 'api_key', retryer=retryer)
```

The setup above would capture any exceptions caused by exceeding the rate limits, and repeatedly retry, waiting an exponentially increasing time between attempts - until successful. Note that any historical data allowance rate limit would not be re-attempted.

See the integration and unit test for examples, and the tenacity docs for more options

The final option, is to initialise the `IGService` class with `use_rate_limiter=True`, ideally with tenacity as well:

```python
from trading_ig.rest import IGService, ApiExceededException
from tenacity import Retrying, wait_exponential, retry_if_exception_type

retryer = Retrying(wait=wait_exponential(),
    retry=retry_if_exception_type(ApiExceededException))
ig_service = IGService('username', 'password', 'api_key', retryer=retryer, use_rate_
↪limiter=True)
```

The rate limiter queries the API for the request limits associated with the API key you logged in with when the session is created.

There are four limit types defined by the API:

allowanceAccountHistoricalData	Historical price data data points per minute allowance
allowanceAccountOverall	Per account request per minute allowance
allowanceAccountTrading	Per account trading request per minute allowance
allowanceApplicationOverall	Overall request per minute allowance

The limits are different between demo and live, live being less restrictive.

The rate limiter does not use allowanceApplicationOverall since this applies accross multiple API logins. It also does not use allowanceAccountHistoricalData becuase this has not yet been implemented.

allowanceAccountOverall is used to set the rate for non-trading requests. allowanceAccountTrading is used to set the rate for trading requests.

The rate limiter actually uses the published values per minute less two, largely to account for the session refresh overhead which happens every 60 seconds. You may still see some 403 errors, but it should be a lot less.

The rate limiter uses these values to effectively release a token for each rate at the required interval. Methods which make requests will block briefly waiting for a new token to be released for the associated rate limit.

When the rate limiter is enabled, the number of requests sent and availble per minute is shown by the logging.

The rate limiter functionality uses threads which exit when `IGService.logout()` is called, so it is important to ensure log out happens or these threads will be left spinning until `__del__()` cleans them up.

6.2.8 Why do see an error like `REJECT_CFD_ORDER_ON_SPREADBET_ACCOUNT`?

If you are attempting to open a spread bet OTC position with code like

```
>>> resp = ig_service.create_open_position(
    currency_code='GBP',
    direction='BUY',
    epic='CS.D.USCGC.TODAY.IP',
    order_type='MARKET',
    expiry='-',
    force_open='false',
    ...
```

you will see this error. CFD bets should have:

```
expiry='-'
```

but spread bets must have:

```
expiry='DFB'
```

or, for futures or forward bets, something like:

```
expiry='SEP-21'
```

6.2.9 Why does my Lightstreamer connection fail after 2 hours / every day?

This problem has come up many times, and there is not really a good solution yet. Have a look at the discussions in these issues:

- Issue 84
- Issue 182

Contributions welcome!

6.2.10 Why do see an error like `public-api.failure.stockbroking-not-supported`?

With the v1 and v2 session endpoints, you only need to specify a username, password and API key to create a session. The APIs only work with spread bet and CFD accounts, but IG offer all sorts of other accounts, eg ISA, SIPP, share trading etc. As a result, IG defines a *default account* for you, which you can change in preferences (or with the API). You will see this error if your default account is set to ISA, SIPP or share trading, and you attempt to login to the API with a v1 or v2 session. There are two solutions:

- change your default account to your spread bet or CFD account. From the web interface, go
  - *My IG > Settings > Default view*
- switch to v3 sessions. see *here*

6.2.11 How do I check my PR will pass CI checks?

This project uses some automated continuous integration (CI) processes whenever any code is committed, or if someone creates a PR. There are unit tests, code formatting with `black`, and linting with `flake8`. In addition, an integration test gets executed every night. The integration test takes a long time due to the *rate limits*. Before making a PR, please make sure the tests pass - PRs will be rejected if they do not. For code formatting:

```
$ poetry run black .
```

and for linting:

```
$ poetry run flake8 trading_ig docs sample tests
```

for unit tests:

```
$ poetry run pytest --ignore=tests/test_integration.py
```

for integration tests:

```
$ poetry run pytest tests/test_integration.py
```

for unit and integration tests:

```
$ poetry run pytest
```

for all tests, including one *really* long running one that tests v3 sessions:

```
$ poetry run pytest --runslow
```

6.2.12 Should I use v2 or v3 sessions?

Short answer: stick with v2 if you can.

Longer answer (read the IG guide first): v1 and v2 sessions are much simpler. Tokens from these sessions are initially valid for 6 hours, but then get extended while in use. This means once a session has been authenticated, your app will continue to be able to make requests indefinitely, as long as you make a request every few hours, say. You would only need to re-authenticate if your connection was reset, for example. Once authenticated with one of these sessions, the active account (eg spread bet, CFD) will be the one defined as your *default account*. You can then switch to another account using `switch_account()`, if needed.

v3 sessions (IG calls them `OAuth`, but they are not) are completely different. v3 session tokens expire after 1 minute, which means there is much more work needed under the hood the manage the connection. Internally, this library checks before each request to see if the session needs to be refreshed, or if a new one is needed. The implementation is newish (April 2021) and is relatively untested. With v3 sessions, you specify which account you wish to connect to at the time of `IGService` creation.

There is one use case where you *must* use v3 sessions (at least so far discovered). If you use both IG's L2 Dealer product for buying and selling shares or CFDs, *and* you have an application connecting to the APIs, then your app will need to use v3 sessions. *L2 Dealer* requires your default account to be set to ISA, SIPP, etc.

6.2.13 How do I connect with a v3 session?

With v3 sessions, you must also supply the account you wish to connect to, use the `acc_number` parameter. You also need to specify `version='3'` in the `create_session()` call

```
>>> from trading_ig.rest import IGService
>>> from trading_ig.config import config
>>> ig_service = IGService(
        config.username,
        config.password,
        config.api_key,
        config.acc_type,
        acc_number=config.acc_number)
>>> ig_service.create_session(version='3')
```

6.2.14 What if I have a problem?

If you have a problem using this library, the first thing to do is to try to isolate where the problem is. The IG platform is a complex application, and there are many ways to make mistakes using it. Just because you see an error, it does not necessarily mean there is a problem with this library. If you encounter an issue, you should follow these steps, in order:

1. Check if there a problem with the IG platform. From time to time the IG platform itself has issues, especially the DEMO environment. If you see a message like *ConnectionRefusedError*, or a 500 Server error, then it could be an issue with the IG platform. IG provide a status page, though its accuracy is questionable. You can also check the IG Community forums. If there are platform issues, its likely someone will have already posted a message there.

2. Check if there is a problem with your code. Most of the API endpoints have multiple options, multiple versions, multiple ways of accessing them, and multiple interdependent parameters. Incoming data is validated on the server, and problems will be reported back in the response. You should

- check the documentation (REST, Streaming) to make sure you are calling the APIs correctly
- look at the sample code and unit and integration tests. There are example snippets for most API endpoints.
- repeat the API call using the IG companion tools (REST, Streaming). If you get the same result, then it is likely that you are using the API incorrectly.

Unfortunately, the people who maintain this library do not have time to provide support. In this case you should:

- read the IG docs more carefully, or
- post a question in the IG Community site, or
- contact the API support team

3. If you're sure that the problem is with this library, please:
   - provide *everything* necessary to reproduce the problem
   - include the full script that produces the error, including import statements
   - ideally this should be a *minimal example* - the shortest possible script that reproduces the problem
   - dependencies and their versions
   - the full output trace including the error messages

An issue without all this information may be ignored and/or closed without response

6.2.15 What happened to `setup.py` and `requirements.txt`?

Early versions of this project used the standard `setup.py` config, with a `requirements.txt` file describing dependencies. Poetry support was added with version 0.0.10 (July 2021). The old style config was removed with version 0.0.14

6.2.16 Why are some dependencies marked as optional in `pyproject.toml`?

Short answer: flexibility. Longer answer:

- The original intent of the project was that `pandas` and `munch` usage was optional. At a low level the IG APIs return JSON data in the response body; this project aims to be a flexible as possible in how applications use that data. If your project has pandas available, then the data will be converted into a pandas DataFrame where it makes sense to do so. Time series data for example, like historical price data, or account activity. If not, it returns a dict of the response data. It's the same for munch - fetching market info for a given epic will return a munch object if that library is available in your environment, or a dict if not
- if this project is defined as a dependency in a higher level project (ie as a library), it should not define which version of pandas is used. That should be defined in the parent project
- `tenacity` support was added in version 0.0.10 as one possible way to handle the IG rate limits. However, it is a brute force method, effectively waiting an ever increasing amount of time between attempts until a request succeeds. It works well for the nightly integration test, where time taken is not important, but for high speed trading with real money it may not be be best solution. There are many other ways to handle the limits, and each will depend on the characteristics of the application. To be as flexible as possible for users of this project, tenacity is also marked optional

6.2.17 How do I find the epic for market 'X'?

There are a few different ways:

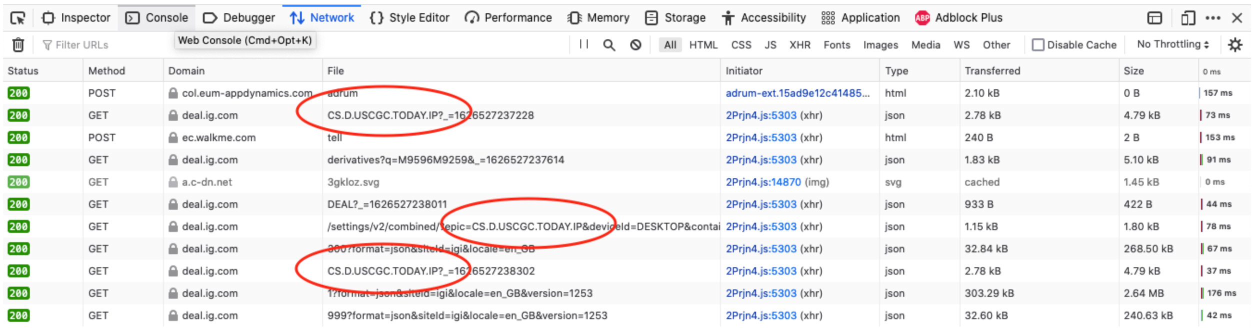
1. Use the REST API Companion. This is a good tool to get familiar with anyway, if you want to learn about the IG APIs. Login, then use the *Search Markets* part, enter your search term, and press *Go*. The results will show any markets that match the search term you entered, and *epic* is one of the attributes displayed in the results. Its a simple text search though, there's no way to filter.

2. Use this library, see method `IGService.search_markets()`. This is the same function that sits behind method 1 above.

3. (Recommended) Use the IG website. Login to the IG site, and show developer tools (Chrome, Firefox) in your browser, with the network tab selected. Internally, the IG website content is driven by a version of the same API, the URLs are similar. So, navigate to the market you want to find the epic for, then check the network tab. The URL will contain the epic, eg for the URL:

```
https://deal.ig.com/nwtpdeal/v2/markets/details/CS.D.USCGC.TODAY.IP?_=1626527237228
```

The epic is *CS.D.USCGC.TODAY.IP*. See screenshot:



Status	Method	Domain	File	Initiator	Type	Transferred	Size	0 ms
200	POST	col.eum-appdynamics.com	adrum	adrum-ext.15ad9e12c41485...	html	2.10 kB	0 B	157 ms
200	GET	deal.ig.com	CS.D.USCGC.TODAY.IP?_=1626527237228	2Prjn4.js:5303 (xhr)	json	2.78 kB	4.79 kB	73 ms
200	POST	ec.walkme.com	tell	2Prjn4.js:5303 (xhr)	html	240 B	2 B	153 ms
200	GET	deal.ig.com	derivatives?q=M9596M9259&_=1626527237614	2Prjn4.js:5303 (xhr)	json	1.83 kB	5.10 kB	91 ms
200	GET	a.c-dn.net	3gkloz.svg	2Prjn4.js:14870 (img)	svg	cached	1.45 kB	0 ms
200	GET	deal.ig.com	DEAL?_=1626527238011	2Prjn4.js:5303 (xhr)	json	933 B	422 B	44 ms
200	GET	deal.ig.com	/settings/v2/combined/?epic=CS.D.USCGC.TODAY.IP&deviceId=DESKTOP&contai	2Prjn4.js:5303 (xhr)	json	1.15 kB	1.80 kB	78 ms
200	GET	deal.ig.com	300?format=json&siteId=igi&locale=en_GB	2Prjn4.js:5303 (xhr)	json	32.84 kB	268.50 kB	67 ms
200	GET	deal.ig.com	CS.D.USCGC.TODAY.IP?_=1626527238302	2Prjn4.js:5303 (xhr)	json	2.78 kB	4.79 kB	37 ms
200	GET	deal.ig.com	1?format=json&siteId=igi&locale=en_GB&version=1253	2Prjn4.js:5303 (xhr)	json	303.29 kB	2.64 MB	176 ms
200	GET	deal.ig.com	999?format=json&siteId=igi&locale=en_GB&version=1253	2Prjn4.js:5303 (xhr)	json	32.60 kB	240.63 kB	42 ms

4. The REST API has two methods that can be used to replicate the navigation tree used on the IG website - `fetch_top_level_navigation_nodes()` and `fetch_sub_nodes_by_node()`. There is also a script in the `/samples` directory that shows how these functions *could* be used to traverse the entire tree. However, this is not recommended; the tree is HUGE, and it would take days to traverse the entire tree due to the rate limits.

6.2.18 Why do see an error like `unauthorised.access.to.equity.exception`?

It is not really clear what this means. It can currently (July 2021) be seen if attempting to get historic prices for the Volatility Index (VIX) futures spread bet. On querying the API support team, their response was:

> Our developers and dealing desk looked into the issue and VIX data should not be available on web api at all. The VIX data feed is similar to the shares datafeed. The error you received on the futures (feb , mar etc) is correct and the cash should also be disabled. Our developers have an existing project to bring the Cash epic in line with the future contracts but going forward you would not be able to download data on any of the VIX epics using API.

6.3 REST API

You can use the IG Markets HTTP / REST API to submit trade orders, open positions, close positions and view market sentiment.

Full details about the API along with information about how to open an account with IG can be found at the link below:

http://labs.ig.com/

6.3.1 How To Use The Library

Using this library to connect to the IG Markets API is extremely easy. All you need to do is import the IGService class, create an instance, and call the methods you wish to use. There is a method for most of the endpoints exposed by their API. The code sample below shows you how to connect to the API, switch to a secondary account and retrieve all open positions for the active account.

```
from trading_ig import IGService
from trading_ig.config import config

ig_service = IGService(config.username, config.password, config.api_key, config.acc_
↪type)
```

(continues on next page)

(continued from previous page)

```
ig_service.create_session()

account_info = ig_service.switch_account(config.acc_number, False) # not necessary
print(account_info)

open_positions = ig_service.fetch_open_positions()
print("open_positions:\n%s" % open_positions)

print("")

epic = 'CS.D.EURUSD.MINI.IP'
resolution = 'D'
num_points = 10
response = ig_service.fetch_historical_prices_by_epic_and_num_points(epic, resolution,
↪ num_points)
df_ask = response['prices']['ask']
print("ask prices:\n%s" % df_ask)
```

with `trading_ig_config.py`

```
class config(object):
    username = "YOUR_USERNAME"
    password = "YOUR_PASSWORD"
    api_key = "YOUR_API_KEY"
    acc_type = "DEMO" # LIVE / DEMO
    acc_number = "ABC123"
```

Config can also be set as environment variable

```
export IG_SERVICE_USERNAME="..."
export IG_SERVICE_PASSWORD="..."
export IG_SERVICE_API_KEY="..."
export IG_SERVICE_ACC_TYPE="DEMO" # LIVE / DEMO
export IG_SERVICE_ACC_NUMBER="..."
```

it should display:

```
open_positions:
Empty DataFrame
Columns: []
Index: []

ask prices:
                        Open     High      Low    Close
DateTime
2014:11:18-00:00:00  1.24510  1.25465  1.24442  1.25330
2014:11:19-00:00:00  1.25332  1.26013  1.25127  1.25461
2014:11:20-00:00:00  1.25463  1.25760  1.25048  1.25427
2014:11:21-00:00:00  1.25428  1.25689  1.23755  1.23924
2014:11:23-00:00:00  1.23640  1.23770  1.23607  1.23725
2014:11:24-00:00:00  1.23864  1.24453  1.23830  1.24390
2014:11:25-00:00:00  1.24389  1.24877  1.24026  1.24743
2014:11:26-00:00:00  1.24744  1.25322  1.24443  1.25077
2014:11:27-00:00:00  1.25078  1.25244  1.24569  1.24599
2014:11:28-00:00:00  1.24598  1.24909  1.24269  1.24505
```

Many IGService methods return Python Pandas DataFrame, Series or Panel

Cache queries requests-cache

Set CachedSession using:

```
from datetime import datetime, timedelta
import requests_cache
session = requests_cache.CachedSession(cache_name='cache', backend='sqlite', expire_
↪after=timedelta(hours=1))
# set expire_after=None if you don't want cache expiration
# set expire_after=0 if you don't want to cache queries
```

CachedSession can be applied globally on IGService

```
ig_service = IGService(config.username, config.password, config.api_key, config.acc_
↪type, session)
ig_service.create_session()
```

or just for a given method (like fetching prices)

```
epic = 'CS.D.EURUSD.MINI.IP'
resolution = 'D'
start_date = '2014-12-15'
end_date = '2014-12-20'
response = ig_service.fetch_historical_prices_by_epic_and_date_range(epic, resolution,
↪ start_date, end_date, session)
```

6.4 Streaming API

You can use the IG Markets STREAM API to get realtime price, balance. . .

Full details about the API along with information about how to open an account with IG can be found at the link below:

http://labs.ig.com/

6.4.1 How To Use The Library

Run sample using:

```
$ python sample/stream_ig.py
INFO:requests.packages.urllib3.connectionpool:Starting new HTTPS connection (1): demo-
↪api.ig.com
INFO:ig_stream:Starting connection with https://demo-apd.marketdatasystems.com
L1:CS.D.USDJPY.CFD.IP: Time 20:35:43 - Bid 119.870 - Ask 119.885
L1:CS.D.GBPUSD.CFD.IP: Time 20:35:46 - Bid 1.51270 - Ask 1.51290
----------HIT CR TO UNSUBSCRIBE AND DISCONNECT FROM     LIGHTSTREAMER-----------
L1:CS.D.USDJPY.CFD.IP: Time 20:35:43 - Bid 119.870 - Ask 119.885
L1:CS.D.USDJPY.CFD.IP: Time 20:35:48 - Bid 119.871 - Ask 119.886
L1:CS.D.GBPUSD.CFD.IP: Time 20:35:48 - Bid 1.51271 - Ask 1.51291
L1:CS.D.USDJPY.CFD.IP: Time 20:35:48 - Bid 119.870 - Ask 119.885
L1:CS.D.GBPUSD.CFD.IP: Time 20:35:49 - Bid 1.51270 - Ask 1.51290

INFO:lightstreamer:Unsubscribed successfully
WARNING:lightstreamer:Server error
DISCONNECTED FROM LIGHTSTREAMER
```

CHAPTER 7

Credits

Created by Femto Trader, Lewis Barber, Sudipta Basak, Andy Geach

Maintained by Andy Geach