TradingBot Documentation

Release 1.0.0

Alberto Cardellini

May 09, 2019

CONTENTS

1 Introduction **1**
- 1.1 System Overview . . . 1
- 1.2 Modules . . . 3
- 1.3 Changelog . . . 10

2 TradingBot **11**

3 Dependencies **13**

4 Install **15**

5 Setup **17**
- 5.1 Market source . . . 17
- 5.2 Configuration file . . . 18

6 Start TradingBot **21**
- 6.1 Close all the open positions . . . 21

7 Stop TradingBot **23**

8 Test **25**

9 Documentation **27**

10 Automate **29**

11 Docker **31**

12 Contributing **33**

Python Module Index **35**

CHAPTER

ONE

INTRODUCTION

TradingBot is an autonomous trading system that uses customised strategies to trade in the London Stock Exchange market. This documentation provides an overview of the system, explaining how to create new trading strategies and how to integrate them with TradingBot. Explore the next sections for a detailed documentation of each module too.

1.1 System Overview

TradingBot is a python script with the goal to automate the trading of stocks in the London Stock Exchange market. It is designed around the idea that to trade in the stock market you need a **strategy**: a strategy is a set of rules that define the conditions where to buy, sell or hold a certain market. TradingBot design lets the user implement a custom strategy without the trouble of developing all the boring stuff to make it work.

The following sections give an overview of the main components that compose TradingBot.

1.1.1 TradingBot

TradingBot is the main entiy used to initialised all the components that will be used during the main routine. It reads the configuration file and the credentials file, it creates the configured strategy instance, the broker interface and it handle the processing of the markets with the active strategy.

1.1.2 Broker interface

TradingBot requires an interface with an executive broker in order to open and close trades in the market. The broker interface is initialised in the `TradingBot` module and it should be independent from its underlying implementation.

At the current status, the only supported broker is IGIndex. This broker provides a very good set of API to analyse the market and manage the account. TradingBot makes also use of other 3rd party services to fetch market data such as price snapshot or technical indicators.

1.1.3 Strategy

The `Strategy` is the core of the TradingBot system. It is a generic template class that can be extended with custom functions to execute trades according to the personalised strategy.

How to use your own strategy

Anyone can create a new strategy from scratch in a few simple steps. With your own strategy you can define your own set of rules to decide whether to buy, sell or hold a specific market.

1. Create a new python module inside the Strategy folder :

```
cd Strategies
touch my_strategy.py
```

2. Edit the file and add a basic strategy template like the following:

```
import os
import inspect
import sys
import logging

# Required for correct import path
currentdir = os.path.dirname(os.path.abspath(inspect.getfile(inspect.
↪currentframe())))
parentdir = os.path.dirname(currentdir)
sys.path.insert(0,parentdir)

from .Strategy import Strategy
from Utils import Utils, TradeDirection
# Import any other required module

class my_strategy(Strategy): # Extends Strategy module
    def __init__(self, config, broker):
        # Call parent constructor
        super().__init__(config, broker)

    def read_configuration(self, config):
        # Read from the config json and store config parameters

    def find_trade_signal(self, epic_id):
        # Given an IG epic decide the trade direction
        # Here is where you want to implement your own code!
        # return TradeDirection.XXX, stop_level, limit_level

    def get_seconds_to_next_spin(self):
        # Return the amount of seconds between each spin of the strategy
        # Each spin analyses all the markets in a list/watchlist
```

3. Add the implementation for these functions:
   - *read_configuration*: `config` is the json object loaded from the `config.json` file
   - *find_trade_signal*: it is the core of your custom strategy, here you can use the broker interface to decide if trade the given epic
   - *get_seconds_to_next_spin*: the *find_trade_signal* is called for every epic requested. After that TradingBot will wait for the amount of seconds defined in this function
4. `Strategy` parent class provides a `Broker` type internal member that can be accessed with `self.broker`. This member is the TradingBot broker interface and provide functions to fetch market data, historic prices and technical indicators. See the *Modules* section for more details.
5. Edit the `StrategyFactory` module inporting the new strategy and adding its name to the `StrategyNames` enum. Then add it to the *make* function

```
28    def make_strategy(self, strategy_name):
29        if strategy_name == StrategyNames.SIMPLE_MACD.value:
30            return SimpleMACD(self.config, self.broker)
```

(continues on next page)

(continued from previous page)

```
31          elif strategy_name == StrategyNames.FAIG.value:
32              return FAIG_iqr(self.config, self.broker)
33          elif strategy.name == StrateyNames.MY_STRATEGY.value:
34              return MY_STRATEGY(self.config, self.broker)
35          else:
36              logging.error('Impossible to create strategy {}. It does not exist'.
↪format(strategy_name))
```

6. Edit the `config.json` adding a new section for your strategy parameters
7. Create a unit test for your strategy
8. Share your strategy creating a Pull Request in GitHub :)

1.2 Modules

TradingBot is composed by different modules organised by their nature. Each section of this document provide a description of the module meaning along with the documentation of its internal members.

1.2.1 TradingBot

class TradingBot.**TradingBot**

Class that initialise and hold references of main components like the broker interface, the strategy or the epic_ids list

close_open_positions()

Closes all the open positions in the account

init_trading_services(*config*, *credentials*)

Create instances of the trading services required, such as web interface for trading and fetch market data.

- **config** The configuration json
- **credentials** The credentials json
- return: An instance of Broker class initialised

load_epic_ids_from_local_file(*filepath*)

Read a file from filesystem containing a list of epic ids. The filepath is defined in config.json file Returns a 'list' of strings where each string is a market epic

load_json_file(*filepath*)

Load a JSON formatted file from the given filepath

- **filepath** The filepath including filename and extension
- Return a dictionary of the loaded json

process_epic_list(*epic_list*)

Process the given list of epic ids, one by one to find new trades

- **epic_list**: list of epic ids as strings

process_market(*epic*)

Process the givem epic using the defined strategy

- **epic**: string representing a market epic id
- Returns **False** if market is closed or if account reach maximum margin, otherwise **True**

process_market_exploration(*node_id*)
Navigate the markets using IG API to fetch markets id dinamically

- **node_id**: The node id to navigate markets in

process_open_positions(*positions*)
process the open positions to find closing trades

- **positions**: json object containing open positions
- Returns **False** if an error occurs otherwise True

process_trade(*epic*)
Process a trade checking if it is a "close position" trade or a new action

process_watchlist(*watchlist_name*)
Process the markets included in the given IG watchlist

- **watchlist_name**: IG watchlist name

read_configuration(*config*)
Read the configuration from the config json

setup_logging()
Setup the global logging settings

start(*argv*)
Starts the TradingBot

wait_for_next_market_opening()
Sleep until the next market opening. Takes into account weekends and bank holidays in UK

1.2.2 Interfaces

The `Interfaces` module contains all those interfaces with external services used by TradingBot. The `Broker` class is the wrapper of all the trading services and provides the main interface for the `strategies` to access market data and perform trades.

IGInterface

class Interfaces.IGInterface.**IGInterface**(*config*, *credentials*)
IG broker interface class, provides functions to use the IG REST API

authenticate(*credentials*)
Authenticate the IGInterface instance with the given credentials

- **credentials**: json object containing username, passowrd, default account and api key
- Returns **False** if an error occurs otherwise True

close_all_positions()
Try to close all the account open positions.

- Returns **False** if an error occurs otherwise True

close_position(*position*)
Close the given market position

- **position**: position json object obtained from IG API
- Returns **False** if an error occurs otherwise True

confirm_order(*dealRef*)
Confirm an order from a dealing reference

- **dealRef**: dealing reference to confirm
- Returns **False** if an error occurs otherwise True

get_account_balances()
Returns a tuple (balance, deposit) for the account in use

- Returns **(None,None)** if an error occurs otherwise (balance, deposit)

get_account_used_perc()
Fetch the percentage of available balance is currently used

- Returns the percentage of account used over total available amount

get_market_info(*epic_id*)
Returns info for the given market including a price snapshot

- **epic_id**: market epic as string
- Returns **None** if an error occurs otherwise the json returned by IG API

get_markets_from_watchlist(*name*)
Get the list of markets included in the watchlist

- **name**: name of the watchlist

get_open_positions()
Returns the account open positions in an json object

- Returns the json object returned by the IG API

get_positions_map()
Returns a *dict* containing the account open positions in the form {string: int} where the string is defined as 'marketId-tradeDirection' and the int is the trade size

- Returns **None** if an error occurs otherwise a dict(string:int)

get_prices(*epic_id*, *interval*, *data_range*)
Returns past prices for the given epic

- **epic_id**: market epic as string
- **interval**: resolution of the time series: minute, hours, etc.
- **data_range**: amount of datapoint to fetch
- Returns **None** if an error occurs otherwise the json object returned by IG API

get_watchlist(*id*)
Get the watchlist info

- **id**: id of the watchlist. If empty id is provided, the function returns the list of all the watchlist in the account

http_get(*url*)
Perform an HTTP GET request to the url. Return the json object returned from the API if 200 is received Return None if an error is received from the API

macd_dataframe(*epic*, *interval*)
Return a datafram with MACD data for the requested market

navigate_market_node(*node_id*)
Navigate the market node id

- Returns the json representing the market node

read_configuration(*config*)
Read the configuration from the config json

set_default_account(*accountId*)
Sets the IG account to use

- **accountId**: String representing the accound id to use
- Returns **False** if an error occurs otherwise True

trade(*epic_id*, *trade_direction*, *limit*, *stop*)
Try to open a new trade for the given epic

- **epic_id**: market epic as string
- **trade_direction**: BUY or SELL
- **limit**: limit level
- **stop**: stop level
- Returns **False** if an error occurs otherwise True

AVInterface

class Interfaces.AVInterface.**AVInterface**(*apiKey*, *config*)
AlphaVantage interface class, provides methods to call AlphaVantage API and return the result in useful format handling possible errors.

daily(*marketId*)
Calls AlphaVantage API and return the Daily time series for the given market

- **marketId**: string representing an AlphaVantage compatible market id
- Returns **None** if an error occurs otherwise the pandas dataframe

get_prices(*market_id*, *interval*)
Return the price time series of the requested market with the interval granularity. Return None if the interval is invalid

intraday(*marketId*, *interval*)
Calls AlphaVantage API and return the Intraday time series for the given market

- **marketId**: string representing an AlphaVantage compatible market id
- **interval**: string representing an AlphaVantage interval type
- Returns **None** if an error occurs otherwise the pandas dataframe

macd(*marketId*, *interval*)
Calls AlphaVantage API and return the MACDEXT tech indicator series for the given market

- **marketId**: string representing an AlphaVantage compatible market id
- **interval**: string representing an AlphaVantage interval type
- Returns **None** if an error occurs otherwise the pandas dataframe

macdext(*marketId*, *interval*)
Calls AlphaVantage API and return the MACDEXT tech indicator series for the given market

- **marketId**: string representing an AlphaVantage compatible market id
- **interval**: string representing an AlphaVantage interval type

- Returns **None** if an error occurs otherwise the pandas dataframe

quote_endpoint(*market_id*)
Calls AlphaVantage API and return the Quote Endpoint data for the given market

- **market_id**: string representing the market id to fetch data of
- Returns **None** if an error occurs otherwise the pandas dataframe

weekly(*marketId*)
Calls AlphaVantage API and return the Weekly time series for the given market

- **marketId**: string representing an AlphaVantage compatible market id
- Returns **None** if an error occurs otherwise the pandas dataframe

Broker

class Interfaces.Broker.**Broker**(*config*, *services*)
This class provides a template interface for all those broker related actions/tasks wrapping the actual implementation class internally

close_all_positions()
IG INDEX API ONLY Attempt to close all the current open positions

close_position(*position*)
IG INDEX API ONLY Attempt to close the requested open position

get_account_used_perc()
IG INDEX API ONLY Returns the account used value in percentage

get_market_from_watchlist(*watchlist_name*)
IG INDEX API ONLY Return a name list of the markets in the required watchlist

get_market_info(*epic*)
IG INDEX API ONLY Return the last available snapshot of the requested market as a dict: - data = {'market_id': <value>, 'bid': <value>,'offer': <value>, 'stop_distance_min': <value>}

get_open_positions()
IG INDEX API ONLY Returns the current open positions

get_prices(*epic*, *market_id*, *interval*, *data_range*)

Return historic price of the requested market as a dictionary:

- data = {'high': [], 'low': [], 'close': [], 'volume': []}

macd_dataframe(*epic*, *market_id*, *interval*)
Return a pandas dataframe containing MACD technical indicator for the requested market with requested interval

navigate_market_node(*node_id*)
IG INDEX API ONLY Return the children nodes of the requested node

to_av_interval(*interval*)
Convert the Broker Interval to AlphaVantage compatible intervals. Return the converted interval or None if a conversion is not available

trade(*epic*, *trade_direction*, *limit*, *stop*)
IG INDEX API ONLY Request a trade of the given market

1.2.3 Strategies

The `Strategies` module contains the strategies used by TradingBot to analyse the markets. The `Strategy` class is the parent from where any custom strategy **must** inherit from. The other modules described here are strategies available in TradingBot.

Strategy

class Strategies.Strategy.**Strategy**(*config*, *broker*)

Generic strategy template to use as a parent class for custom strategies. Provide safety checks for new trades and handling of open positions.

find_trade_signal(*epic_id*)

Must override

get_seconds_to_next_spin()

Must override

read_configuration(*config*)

Must override

StrategyFactory

class Strategies.StrategyFactory.**StrategyFactory**(*config*, *broker*)

Factory class to create instances of Strategies. The class provide an interface to instantiate new objects of a given Strategy name

make_strategy(*strategy_name*)

Create and return an instance of the Strategy class specified by the strategy_name

- **strategy_name**: name of the strategy as defined in the json config file
- Returns an instance of the requested Strategy or None if an error occurres

SimpleMACD

class Strategies.SimpleMACD.**SimpleMACD**(*config*, *broker*)

Strategy that use the MACD technical indicator of a market to decide whether to buy, sell or hold. Buy when the MACD cross over the MACD signal. Sell when the MACD cross below the MACD signal.

calculate_stop_limit(*tradeDirection*, *current_offer*, *current_bid*, *limit_perc*, *stop_perc*)

Calculate the stop and limit levels from the given percentages

find_trade_signal(*epic_id*)

Calculate the MACD of the previous days and find a cross between MACD and MACD signal

- **epic_id**: market epic as string
- Returns TradeDirection, limit_level, stop_level or TradeDirection.NONE, None, None

get_seconds_to_next_spin()

Calculate the amount of seconds to wait for between each strategy spin

read_configuration(*config*)

Read the json configuration

Weighted Average Peak Detection

class Strategies.WeightedAvgPeak.**WeightedAvgPeak**(*config*, *broker*)
All credits of this strategy goes to GitHub user @tg12.

find_trade_signal(*epic_id*)
TODO add description of strategy key points

get_seconds_to_next_spin()
Must override

peakdet(*v*, *delta*, *x=None*)
Converted from MATLAB script at http://billauer.co.il/peakdet.html

Returns two arrays

function [maxtab, mintab]=peakdet(v, delta, x) %PEAKDET Detect peaks in a vector % [MAXTAB, MINTAB] = PEAKDET(V, DELTA) finds the local % maxima and minima ("peaks") in the vector V. % MAXTAB and MINTAB consists of two columns. Column 1 % contains indices in V, and column 2 the found values. % % With [MAXTAB, MINTAB] = PEAKDET(V, DELTA, X) the indices % in MAXTAB and MINTAB are replaced with the corresponding % X-values. % % A point is considered a maximum peak if it has the maximal % value, and was preceded (to the left) by a value lower by % DELTA.

% Eli Billauer, 3.4.05 (Explicitly not copyrighted). % This function is released to the public domain; Any use is allowed.

read_configuration(*config*)
Read the json configuration

weighted_avg_and_std(*values*, *weights*)
Return the weighted average and standard deviation.

values, weights – Numpy ndarrays with the same shape.

1.2.4 Utils

class Utils.**Utils**
Utility class containing static methods to perform simple general actions

static get_seconds_to_market_opening(*from_time*)
Return the amount of seconds from now to the next market opening, taking into account UK bank holidays and weekends

static humanize_time(*secs*)
Convert the given time (in seconds) into a readable format hh:mm:ss

static is_between(*time*, *time_range*)
Return True if time is between the time_range. time must be a string. time_range must be a tuple (a,b) where a and b are strings in format 'HH:MM'

static is_market_open(*timezone*)
Return True if the market is open, false otherwise

- **timezone**: string representing the timezone

static midpoint(*p1*, *p2*)
Return the midpoint

static percentage(*part*, *whole*)
Return the percentage value of the part on the whole

static percentage_of(*percent*, *whole*)
Return the value of the percentage on the whole

1.3 Changelog

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog, and this project adheres to Semantic Versioning.

1.3.1 [1.0.0] - 2019-04-21

Added

- Initial release

CHAPTER

TWO

TRADINGBOT

This is an attempt to create an autonomous market trading script using the IG REST API and any other available data source for market prices.

TradingBot is meant to be a "forever running" process that keeps analysing the markets and taking actions whether the conditions are met. It is halfway from an academic project and a real useful piece of software, I guess I will see how it goes :)

The main goal of this project is to provide the capability to write a custom trading strategy with the minimum effort. TradingBot handle all the boring stuff.

All the credits for the FAIG_iqr strategy goes to GitHub user @tg12 who is the creator of the first script version and gave me a good starting point for this project. Thank you.

CHAPTER

THREE

DEPENDENCIES

- Python 3.4+

View file `requirements.txt` for the full list of dependencies.

CHAPTER

FOUR

INSTALL

TradingBot can be controlled by the `trading_bot_ctl` shell script which provides several commands to perform different actions. After cloning this repo, to install TradingBot simply run:

```
sudo ./trading_bot_ctl install
```

The required dependencies will be installed and all necessary files installed in `/opt/TradingBot` by default. It is recommended to add this path to your `PATH` environment variable.

The last step is to set file permissions on the installed folders for your user with the following command:

```
sudo chown -R $USER: $HOME/.TradingBot
```

CHAPTER

FIVE

SETUP

Login to your IG Dashboard

- Obtain an API KEY from the settings panel
- If using the demo account, create demo credentials
- Take note of your spread betting account ID (demo or real)
- Visit AlphaVantage website: `https://www.alphavantage.co`
- Request a free api key
- Insert these info in a file called `.credentials`

This must be in json format

```
{
    "username": "username",
    "password": "password",
    "api_key": "apikey",
    "account_id": "accountId",
    "av_api_key": "apiKey"
}
```

- Copy the `.credentials` file in the `data` folder
- Revoke permissions to read the file if you are paranoid .. code-block:: guess

      cd data sudo chmod 600 .credentials

5.1 Market source

There are different ways to define which markets to analyse with TradinbgBot. You can select your preferred option in the `config.json` file with the `market_source` parameter:

- **Local file**

You can create a file `epic_ids.txt` containg IG epics of the companies you want to monitor. You need to copy this file into the `data` folder.

- **Watchlist**

You can use an IG watchlist, TradingBot will analyse every market added to the selected watchlist

- **API**

TradingBot navigates the IG markets dynamically using the available API call to fetch epic ids.

5.2 Configuration file

The `config.json` file is in the `config` folder and it contains several configurable parameter to personalise how TradingBot work. These are the description of each parameter:

5.2.1 General

- **max_account_usable**: The maximum percentage of account funds to use (A safe value is around 50%)
- **time_zone**: The timezone to use (i.e. 'Europe/London)
- **enable_log**: Enable the log in a file rather than on stdout
- **log_file**: Define the full file path for the log file to use, if enabled. {home} and {timestamp} placeholders are replaced with the user home directory and the timestamp when TradingBot started
- **debug_log**: Enable the debug level in the logging
- **credentials_filepath**: Filepath for the `.credentials` file
- **market_source**: The source to use to fetch the market ids. Available values are explained in the `Setup` section below.
- **epic_ids_filepath**: The full file path for the local file containing the list of epic ids
- **watchlist_name**: The watchlist name to use as market source, if selected
- **active_strategy**: The strategy name to use. Must match one of the names in the `Strategies` section below

5.2.2 IG Interface

- **order_type**: The IG order type (MARKET, LIMIT, etc.). Do NOT change it
- **order_size**: The size of the spread bets
- **order_expiry**: The order expiry (DFB). Do NOT change it
- **order_currency**: The currency of the order (For UK shares leave it as GBP)
- **order_force_open**: Force to open the orders
- **use_g_stop**: Use guaranteed stops. Read IG terms for more info about them.
- **use_demo_account**: Trade on the DEMO IG account. If enabled remember to setup the demo account credentials too
- **controlled_risk**: Enable the controlled risk stop loss calculation. Enable only if you have a controlled risk account.
- **paper_trading**: Enable the `paper trading`. No real trades will be done on the IG account.

5.2.3 Alpha Vantage

- **enable**: Enable the use of AlphaVantage API
- **api_timeout**: Timeout in seconds between each API call

5.2.4 Strategies

Settings specific for each strategy

5.2.5 SimpleMACD

- **spin_interval**: Override the `Strategies` value
- **max_spread_perc**: Spread percentage to filter markets with high spread
- **limit_perc**: Limit percentage to take profit for each trade
- **stop_perc**: Stop percentage to stop any loss

CHAPTER

SIX

START TRADINGBOT

```
./trading_bot_ctl start
```

6.1 Close all the open positions

```
./trading_bot_ctl close_positions
```

CHAPTER

SEVEN

STOP TRADINGBOT

```
./trading_bot_ctl stop
```

CHAPTER

EIGHT

TEST

If you have setup a virtual environment you can run the test by running `pytest` from the project root folder.

You can run the test from a clean environment with:

```
./trading_bot_ctl test
```

You can run the test in Docker containers against different python versions:

```
./trading_bot_ctl test_docker
```

CHAPTER

NINE

DOCUMENTATION

The Sphinx documentation contains further details about each TradingBot module with source code documentation of each class member. Explanation is provided regarding how to create your own Strategy and how to integrate it with the system.

Read the documentation at:

https://tradingbot.readthedocs.io

You can build it locally with:

```
./trading_bot_ctl docs
```

The generated html files will be in `doc/_build/html`.

CHAPTER

TEN

AUTOMATE

NOTE: TradingBot monitors the market opening hours and suspend the trading when the market is closed. Generally you should NOT need a cron job!

You can set up the crontab job to run and kill TradinBot at specific times. The only configuration required is to edit the crontab file adding the preferred schedule:

```
crontab -e
```

As an example this will start TradingBot at 8:00 in the morning and will stop it at 16:35 in the afternoon, every week day (Mon to Fri):

```
00 08 * * 1-5 /.../TradingBot/trading_bot_ctl start
35 16 * * 1-5 /.../TradingBot/trading_bot_ctl stop
```

NOTE: Remember to set the correct timezone in your machine!

CHAPTER

ELEVEN

DOCKER

You can run TradingBot in a Docker container (https://docs.docker.com/):

```
./trading_bot_ctl start_docker
```

The container will be called `dkr_trading_bot` and the logs will still be stored in the configured folder in the host machine. By default `~/.TradingBot/log`.

To stop TradingBot:

```
./trading_bot_ctl stop_docker
```

or just kill the container:

```
docker kill dkr_trading_bot
```

If you need to start a bash shell into the container

```
docker exec -it dkr_trading_bot bash
```

CHAPTER

TWELVE

CONTRIBUTING

I am really happy to receive any help so please just open a pull request with your changes and I will handle it.

If you instead find problems or have ideas for future improvements open an Issue. Thanks for all the support!

PYTHON MODULE INDEX

i

Interfaces.AVInterface, 6
Interfaces.Broker, 7
Interfaces.IGInterface, 4

s

Strategies.SimpleMACD, 8
Strategies.Strategy, 8
Strategies.StrategyFactory, 8
Strategies.WeightedAvgPeak, 9

t

TradingBot, 3

u

Utils, 9

INDEX

A

authenticate() (*Interfaces.IGInterface.IGInterface method*), 4
AVInterface (*class in Interfaces.AVInterface*), 6

B

Broker (*class in Interfaces.Broker*), 7

C

calculate_stop_limit() (*Strategies.SimpleMACD.SimpleMACD method*), 8
close_all_positions() (*Interfaces.Broker.Broker method*), 7
close_all_positions() (*Interfaces.IGInterface.IGInterface method*), 4
close_open_positions() (*TradingBot.TradingBot method*), 3
close_position() (*Interfaces.Broker.Broker method*), 7
close_position() (*Interfaces.IGInterface.IGInterface method*), 4
confirm_order() (*Interfaces.IGInterface.IGInterface method*), 4

D

daily() (*Interfaces.AVInterface.AVInterface method*), 6

F

find_trade_signal() (*Strategies.SimpleMACD.SimpleMACD method*), 8
find_trade_signal() (*Strategies.Strategy.Strategy method*), 8
find_trade_signal() (*Strategies.WeightedAvgPeak.WeightedAvgPeak method*), 9

G

get_account_balances() (*Interfaces.IGInterface.IGInterface method*), 5
get_account_used_perc() (*Interfaces.Broker.Broker method*), 7
get_account_used_perc() (*Interfaces.IGInterface.IGInterface method*), 5
get_market_from_watchlist() (*Interfaces.Broker.Broker method*), 7
get_market_info() (*Interfaces.Broker.Broker method*), 7
get_market_info() (*Interfaces.IGInterface.IGInterface method*), 5
get_markets_from_watchlist() (*Interfaces.IGInterface.IGInterface method*), 5
get_open_positions() (*Interfaces.Broker.Broker method*), 7
get_open_positions() (*Interfaces.IGInterface.IGInterface method*), 5
get_positions_map() (*Interfaces.IGInterface.IGInterface method*), 5
get_prices() (*Interfaces.AVInterface.AVInterface method*), 6
get_prices() (*Interfaces.Broker.Broker method*), 7
get_prices() (*Interfaces.IGInterface.IGInterface method*), 5
get_seconds_to_market_opening() (*Utils.Utils static method*), 9
get_seconds_to_next_spin() (*Strategies.SimpleMACD.SimpleMACD method*), 8
get_seconds_to_next_spin() (*Strategies.Strategy.Strategy method*), 8
get_seconds_to_next_spin() (*Strategies.WeightedAvgPeak.WeightedAvgPeak method*), 9
get_watchlist() (*Interfaces.IGInterface.IGInterface method*), 5

H

http_get() (*Interfaces.IGInterface.IGInterface method*), 5
humanize_time() (*Utils.Utils static method*), 9

I

IGInterface (*class in Interfaces.IGInterface*), 4
init_trading_services() (*TradingBot.TradingBot method*), 3
Interfaces.AVInterface (*module*), 6
Interfaces.Broker (*module*), 7
Interfaces.IGInterface (*module*), 4
intraday() (*Interfaces.AVInterface.AVInterface method*), 6
is_between() (*Utils.Utils static method*), 9
is_market_open() (*Utils.Utils static method*), 9

L

load_epic_ids_from_local_file() (*TradingBot.TradingBot method*), 3
load_json_file() (*TradingBot.TradingBot method*), 3

M

macd() (*Interfaces.AVInterface.AVInterface method*), 6
macd_dataframe() (*Interfaces.Broker.Broker method*), 7
macd_dataframe() (*Interfaces.IGInterface.IGInterface method*), 5
macdext() (*Interfaces.AVInterface.AVInterface method*), 6
make_strategy() (*Strategies.StrategyFactory.StrategyFactory method*), 8
midpoint() (*Utils.Utils static method*), 9

N

navigate_market_node() (*Interfaces.Broker.Broker method*), 7
navigate_market_node() (*Interfaces.IGInterface.IGInterface method*), 5

P

peakdet() (*Strategies.WeightedAvgPeak.WeightedAvgPeak method*), 9
percentage() (*Utils.Utils static method*), 9
percentage_of() (*Utils.Utils static method*), 9
process_epic_list() (*TradingBot.TradingBot method*), 3
process_market() (*TradingBot.TradingBot method*), 3
process_market_exploration() (*TradingBot.TradingBot method*), 3
process_open_positions() (*TradingBot.TradingBot method*), 4
process_trade() (*TradingBot.TradingBot method*), 4
process_watchlist() (*TradingBot.TradingBot method*), 4

Q

quote_endpoint() (*Interfaces.AVInterface.AVInterface method*), 7

R

read_configuration() (*Interfaces.IGInterface.IGInterface method*), 6
read_configuration() (*Strategies.SimpleMACD.SimpleMACD method*), 8
read_configuration() (*Strategies.Strategy.Strategy method*), 8
read_configuration() (*Strategies.WeightedAvgPeak.WeightedAvgPeak method*), 9
read_configuration() (*TradingBot.TradingBot method*), 4

S

set_default_account() (*Interfaces.IGInterface.IGInterface method*), 6
setup_logging() (*TradingBot.TradingBot method*), 4
SimpleMACD (*class in Strategies.SimpleMACD*), 8
start() (*TradingBot.TradingBot method*), 4
Strategies.SimpleMACD (*module*), 8
Strategies.Strategy (*module*), 8
Strategies.StrategyFactory (*module*), 8
Strategies.WeightedAvgPeak (*module*), 9
Strategy (*class in Strategies.Strategy*), 8
StrategyFactory (*class in Strategies.StrategyFactory*), 8

T

to_av_interval() (*Interfaces.Broker.Broker method*), 7
trade() (*Interfaces.Broker.Broker method*), 7
trade() (*Interfaces.IGInterface.IGInterface method*), 6
TradingBot (*class in TradingBot*), 3
TradingBot (*module*), 3

U

Utils (*class in Utils*), 9
Utils (*module*), 9

W

wait_for_next_market_opening() (*TradingBot.TradingBot method*), 4
weekly() (*Interfaces.AVInterface.AVInterface method*), 7
weighted_avg_and_std() (*Strategies.WeightedAvgPeak.WeightedAvgPeak method*), 9

WeightedAvgPeak (*class in Strategies.WeightedAvgPeak*), 9