TradingMate Documentation

Release 0.1.0

Alberto Cardellini

Sep 01, 2022

Contents

CHAPTER 1

Introduction

TradingMate 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 TradingMate. Explore the next sections for a detailed documentation of each module too.

1.1 System Overview

TradingMate is a portfolio manager with the goal to help traders to record their trades and deals in the stock market providing useful statistics and metrics to help backtest and review trade performance. It offers a simple user interface that provide information about the current asset value, the overall profit/loss indicator and the trade history.

1.1.1 TradingMate

TradingMate is the main entiy used to initialised all the components. It is the link between the user interface and the data.

TODO

1.2 Modules

Each section of this document provides the source code documentation of each component of TradingMate.

1.2.1 TradingMate

class tradingmate.**TradingMate**(*config_filepath: pathlib.Path = PosixPath('/home/docs/.TradingMate/config/config.json')*)

Main class that handles the interaction between the User Interface and the underlying business logic of the application

close_view_event() → None
Callback function to handle close event of the user interface

delete_trade_event(*portfolio_id: str*, *trade_id: str*) → None
Callback function to handle delete of a trade

get_app_log_filepath()
Return the full filepath of the log file of application current session

get_portfolios() → List[tradingmate.model.portfolio.Portfolio]
Return the list of active portfolios

get_settings_event()
Callback to handle request to show the settings panel

manual_refresh_event(*portfolio_id: str*) → None
Callback function to handle refresh data request

new_trade_event(*new_trade: tradingmate.model.trade.Trade*, *portfolio_id: str*) → None
Callback function to handle new trade event

open_portfolio_event(*filepath: pathlib.Path*) → None
Callback function to handle request to open a new portfolio file

save_portfolio_event(*portfolio_id: str*, *filepath: pathlib.Path*) → None
Callback function to handle request to save/export the portfolio

save_settings_event(*config*)
Callback to save edited settings

set_auto_refresh(*enabled: bool*, *portfolio_id: str*) → None
Callback function to handle set/unset of auto refresh data

1.2.2 Model

The `model` module contains the business logic and the data management of TradingMate.

Holding

class tradingmate.model.**Holding**(*symbol: str*, *quantity: int*, *open_price: Optional[float] = None*)
Represent a current open position for a Market

add_quantity(*value: int*) → None
Add or subtract (if value is negative) the value to the holding quantity

Portfolio

class tradingmate.model.**Portfolio**(*config: tradingmate.model.configuration.ConfigurationManager*, *trading_log_path: pathlib.Path*)
Represent a trading portfolio including cash available and open market positions. The portfolio is based on a list of trades read from the trading log

add_trade(*new_trade: tradingmate.model.trade.Trade*) → None
Add a new trade into the Portfolio

delete_trade(*trade_id: str*) → None
Remove a trade from the Portfolio

get_cash_available() → float
Return the available cash quantity in the portfolio [int]

get_cash_deposited() → float
Return the amount of cash deposited in the portfolio [int]

get_holding_last_price(*symbol: str*) → Optional[float]
Return the last price for the given symbol

get_holding_list() → List[tradingmate.model.holding.Holding]
Return a list of Holding instances held in the portfolio sorted alphabetically

get_holding_open_price(*symbol: str*) → Optional[float]
Return the last price for the given symbol

get_holding_quantity(*symbol: str*) → int
Return the quantity held for the given symbol

get_holding_symbols() → List[str]
Return a list containing the holding symbols as [string] sorted alphabetically

get_holdings_value() → Optional[float]
Return the value of the holdings held in the portfolio

get_id() → str
Return the portfolio unique id [string]

get_name() → str
Return the portfolio name [string]

get_open_positions_pl() → Optional[float]
Return the sum profit/loss in £ of the current open positions

get_open_positions_pl_perc() → Optional[float]
Return the sum profit/loss in % of the current open positions

get_portfolio_path() → pathlib.Path
Return the complete filepath of the portfolio

get_portfolio_pl() → Optional[float]
Return the profit/loss in £ of the portfolio over the deposited cash

get_portfolio_pl_perc() → Optional[float]
Return the profit/loss in % of the portfolio over deposited cash

get_total_value() → Optional[float]
Return the value of the whole portfolio as cash + holdings

get_trade_history() → List[tradingmate.model.trade.Trade]
Return the trade history as a list

has_unsaved_changes() → bool
Return True if the portfolio has unsaved changes, False othersise

save_portfolio(*filepath: pathlib.Path*) → None
Save the portfolio at the given filepath

DatabaseHandler

class tradingmate.model.**DatabaseHandler**(*config: tradingmate.model.configuration.ConfigurationManager*, *trading_log_path: pathlib.Path*)

Handles the IO operation with the database to handle persistent data

add_trade(*trade: tradingmate.model.trade.Trade*) → None
Add a trade to the database

delete_trade(*trade_id: str*) → None
Remove the trade from the trade history

get_db_filepath() → pathlib.Path
Return the database filepath

get_trades_list() → List[tradingmate.model.trade.Trade]
Return the list of trades stored in the db

get_trading_log_name() → str
Return the trading log database name

read_data(*filepath: pathlib.Path = None*)
Read the trade history from the json database and return the list of trades

- **filepath**: optional, if not set the configured path will be used

write_data(*filepath: pathlib.Path = None*) → bool
Write the trade history to the database

StockPriceGetter

class tradingmate.model.**StockPriceGetter**(*config: tradingmate.model.configuration.ConfigurationManager, update_callback: Callable[[], None]*)

Worker thread that fetches market live prices from an online source

task() → None
The task done by this thread - override in subclasses

ConfigurationManager

class tradingmate.model.**ConfigurationManager**(*config_path: pathlib.Path*)

Class that loads the configuration and credentials files exposing methods to provide the configurable parameters

get_alpha_vantage_api_key() → str
Get the alphavantage api key

get_alpha_vantage_base_url() → str
Get the alphavantage API base URI

get_alpha_vantage_polling_period() → float
Get the alphavantage configured polling period

get_configured_stocks_interface() → str
Get the active configured stock interface

get_credentials_path() → pathlib.Path
Get the filepath of the credentials file

get_editable_config() → Dict[str, Any]
Get a dictionary containing the editable configuration parameters

get_log_filepath() → pathlib.Path
Get the filepath of the log file

get_polling_period() → float
Get the application polling period

get_trading_database_path() → List[pathlib.Path]
Get the filepath of the trading log file

get_yfinance_polling_period() → float
Get the yfinance configured polling period

save_settings(*config: Dict[str, Any]*) → bool
Save the edited configuration settings

Trade

class tradingmate.model.**Trade**(*date: datetime.datetime*, *action: tradingmate.utils.enums.Actions*, *quantity: float*, *symbol: str*, *price: float*, *fee: float*, *sdr: float*, *notes: str*, *id: str = None*)
Represent a trade action

1.2.3 Broker

The `broker` module contains the interfaces to connect to the online market brokers

AlphaVantageInterface

class tradingmate.model.broker.**AlphaVantageInterface**(*config: tradingmate.model.configuration.ConfigurationManager*)
class providing interfaces to request data from AlphaVantage

get_prices(*market_id: str*, *interval: tradingmate.model.broker.alpha_vantage_interface.AVInterval = <AVInterval.DAILY: 'daily'>*) → Optional[Dict[str, Dict[str, str]]]
Return the price time series of the requested market with the interval granularity. Return None if the interval is invalid

YFinanceInterface

class tradingmate.model.broker.**YFinanceInterface**(*config: tradingmate.model.configuration.ConfigurationManager*)

StocksInterface

class tradingmate.model.broker.**StocksInterface**

StocksInterfaceFactory

class tradingmate.model.broker.**StocksInterfaceFactory**(*config: tradingmate.model.configuration.ConfigurationManager*)
Factory for stocks interface class

1.2.4 UI

The `ui` module contains the components that compose the User Interface of TradingMate.

DataInterface

class tradingmate.ui.**DataInterface**(*client: tradingmate.ui.trading_mate_client.TradingMateClient, data_callback: Callable[[List[tradingmate.model.portfolio.Portfolio]], None]*)

Thread that periodically requests the most recent data from TradingMate server notify the parent object through a callback function

task()

The task done by this thread - override in subclasses

TradingMateClient

class tradingmate.ui.**TradingMateClient**(*server: tradingmate.trading_mate.TradingMate*)

Client interface to the TradingMate business logic

get_portfolios() → List[tradingmate.model.portfolio.Portfolio]

Get the loaded portfolios

get_settings_event() → None

Request server to fetch TradingMate settings

is_portfolio_auto_refreshing(*portfolio_id: str*) → bool

Return True if portfolio has data auto refresh enabled, False otherwise

manual_refresh_event(*portfolio_id: str*) → None

Request server to refresh portfolio data

new_trade_event(*new_trade: tradingmate.model.trade.Trade*, *portfolio_id: str*) → None

Push new trade notification to the server

open_portfolio_event(*filepath: pathlib.Path*) → None

Request server to open a portfolio

save_portfolio_event(*portfolio_id: str*, *filepath: pathlib.Path*) → None

Request server to save a portfolio

save_settings_event(*settings: Dict[str, Any]*) → None

Request server to save the settings

set_auto_refresh_event(*value: bool*, *portfolio_id: str*) → None

Set server to automatically update data for the portfolio

stop() → None

Handle stop event

unsaved_changes() → bool

Request if open portfolios have unsaved changes and return True

GTK

The `gtk` module contains the gtk components and widgets of the graphical interface. They are not documented due to a Sphinx issue when importing the `gi` Python module

1.2.5 Utils

The `utils` module contains all the utlity components.

Enums

class tradingmate.utils.**Actions**
An enumeration.

class tradingmate.utils.**Markets**
An enumeration.

class tradingmate.utils.**Messages**
An enumeration.

TaskThread

class tradingmate.utils.**TaskThread**
Thread that executes a task every N seconds

cancel_timeout()
Cancel the timeout and run the task

enable(*enabled: bool*) → None
Disable/enable this thread

run() → None
Method representing the thread's activity.

You may override this method in a subclass. The standard run() method invokes the callable object passed to the object's constructor as the target argument, if any, with sequential and keyword arguments taken from the args and kwargs arguments, respectively.

setInterval(*interval: float*) → None
Set the number of seconds we sleep between executing our task

shutdown() → None
Stop this thread

task() → None
The task done by this thread - override in subclasses

Utils

class tradingmate.utils.**Utils**
Class that provides utility functions

static get_install_path() → str
Returns the installation path of TradingMate

static load_json_file(*filepath: pathlib.Path*) → Any
Load a JSON formatted file from the given filepath

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

static write_json_file(*filepath: pathlib.Path*, *data: Any*) → bool
Write a python dict object into a file with json formatting

-**filepath** The filepath -**data** The python dict to write - Return True if succed, False otherwise

Python Module Index

t

Index

H

I

L

M

N

O

P

R

S

T

U

W

Y