TradingMate Documentation

Release 0.1.0

Alberto Cardellini

Mar 14, 2020

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='/opt/TradingMate/config/config.json'*, *log_filepath='/opt/TradingMate/log/trading_mate_{timestamp}.log'*)

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

close_view_event()
Callback function to handle close event of the user interface

delete_trade_event(*portfolio_id*, *trade_id*)
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()
Return the list of active portfolios

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

manual_refresh_event(*portfolio_id*)
Callback function to handle refresh data request

new_trade_event(*new_trade*, *portfolio_id*)
Callback function to handle new trade event

open_portfolio_event(*filepath*)
Callback function to handle request to open a new portfolio file

save_portfolio_event(*portfolio_id*, *filepath*)
Callback function to handle request to save/export the portfolio

save_settings_event(*config*)
Callback to save edited settings

set_auto_refresh(*enabled*, *portfolio_id*)
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 `Model.Holding.`**Holding**(*symbol*, *quantity*, *open_price=None*)

add_quantity(*value*)
Add or subtract (if value is negative) the value to the holding quantity

Portfolio

class `Model.Portfolio.`**Portfolio**(*config*, *trading_log_path*)

add_trade(*new_trade*)
Add a new trade into the Portfolio

delete_trade(*trade_id*)
Remove a trade from the Portfolio

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

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

get_holding_last_price(*symbol*)
Return the last price for the given symbol

get_holding_list()
Return a list of Holding instances held in the portfolio sorted alphabetically

get_holding_open_price(*symbol*)
Return the last price for the given symbol

get_holding_quantity(*symbol*)
Return the quantity held for the given symbol

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

get_holdings_value()
Return the value of the holdings held in the portfolio

get_id()
Return the portfolio unique id [string]

get_name()
Return the portfolio name [string]

get_open_positions_pl()
Return the sum profit/loss in £ of the current open positions

get_open_positions_pl_perc()
Return the sum profit/loss in % of the current open positions

get_portfolio_path()
Return the complete filepath of the portfolio

get_portfolio_pl()
Return the profit/loss in £ of the portfolio over the deposited cash

get_portfolio_pl_perc()
Return the profit/loss in % of the portfolio over deposited cash

get_total_value()
Return the value of the whole portfolio as cash + holdings

get_trade_history()
Return the trade history as a list

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

save_portfolio(*filepath*)
Save the portfolio at the given filepath

DatabaseHandler

class Model.DatabaseHandler.**DatabaseHandler**(*config*, *trading_log_path*)
Handles the IO operation with the database to handle persistent data

add_trade(*trade*)
Add a trade to the database

delete_trade(*trade_id*)
Remove the trade from the trade history

get_db_filepath()
Return the database filepath

get_trades_list()
Return the list of trades stored in the db

get_trading_log_name()
Return the trading log database name

read_data(*filepath=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=None*)
Write the trade history to the database

StockPriceGetter

class Model.StockPriceGetter.**StockPriceGetter**(*config*, *update_callback*)

task()
The task done by this thread - override in subclasses

1.2.3 UI

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

DataInterface

class UI.DataInterface.**DataInterface**(*client*, *data_callback*)
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 UI.TradingMateClient.**TradingMateClient**(*server*)
Client interface to the TradingMate business logic

get_portfolios()
Get the loaded portfolios

get_settings_event()
Request server to fetch TradingMate settings

is_portfolio_auto_refreshing(*portfolio_id*)
Return True if portfolio has data auto refresh enabled, False otherwise

manual_refresh_event(*portfolio_id*)
Request server to refresh portfolio data

new_trade_event(*new_trade*, *portfolio_id*)
Push new trade notification to the server

open_portfolio_event(*filepath*)
Request server to open a portfolio

save_portfolio_event(*portfolio_id*, *filepath*)
Request server to save a portfolio

save_settings_event(*settings*)
Request server to save the settings

set_auto_refresh_event(*value*, *portfolio_id*)
Set server to automatically update data for the portfolio

stop()
Handle stop event

unsaved_changes()
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

1.2.4 Utils

The `Utils` module contains all the utlity components.

ConfigurationManager

class `Utils.ConfigurationManager.`**ConfigurationManager**(*config_path*)
Class that loads the configuration and credentials json files exposing static methods to provide the configurable parameters

get_alpha_vantage_api_key()
Get the alphavantage api key

get_alpha_vantage_base_url()
Get the alphavantage API base URI

get_alpha_vantage_polling_period()
Get the alphavantage configured polling period

get_configured_stocks_interface()
Get the active configured stock interface

get_credentials_path()
Get the filepath of the credentials file

get_editable_config()
Get a dictionary containing the editable configuration parameters

get_polling_period()
Get the application polling period

get_trading_database_path()
Get the filepath of the trading log file

get_yfinance_polling_period()
Get the yfinance configured polling period

save_settings(*config*)
Save the edited configuration settings

TaskThread

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

cancel_timeout()
Cancel the timeout and run the task

enable(*enabled*)
Disable/enable this thread

run()
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*)
Set the number of seconds we sleep between executing our task

shutdown()
Stop this thread

task()
The task done by this thread - override in subclasses

Trade

class Utils.Trade.**Trade**(*date*, *action*, *quantity*, *symbol*, *price*, *fee*, *sdr*, *notes*, *id=None*)

Utils

class Utils.Utils.**Actions**
An enumeration.

class Utils.Utils.**Messages**
An enumeration.

class Utils.Utils.**Markets**
An enumeration.

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 [2.2.0] 2020-03-14

Fixed

- Bug preventing to add a trade with fee equal to zero

Changed

- Updated Pipfile unifying packages and adding custom scripts
- Grouped icons in the header bar under one single popover menu

Added

- Added tooltips to UI widgets
- Icon in status bar that shows internet connection status
- Show application version in About dialog
- Support for yfinance module to fetch stocks data
- Support adding trades happened in the past
- Added popup menu in trades history treeview with option to add and remove trades
- Explore window to show information and details of single markets
- Time granularity to new trades

Fixed

- Fixed bug where main window was hidden when closing app with unsaved changes

1.3.2 [2.1.1] - 2020-01-13

Changed

- Removed unused resource files
- Updated README

1.3.3 [2.1.0] - 2020-01-12

Changed

- Replaced TK user interface with GTK+ 3
- Tickers prices are fetched using `alpha-vantage` Python module
- **alpha_vantage_polling_period** configuration parameter is used to wait between each AV call
- AlphaVantage http requests are thread safe

Added

- Status bar showing portfolio filepath
- Button to open a new window tailing the current application log file

1.3.4 [2.0.0] - 2019-12-14

Changed

- Issue37 - Improved installation process and dependencies setup
- Updated default .credentials configured path
- Re-design of system architecture and API
- Edited Portfolios are not saved automatically and a warning is displayed

Added

- Added Pipfile to manage python dependencies
- Added `FEE` action
- Added `notes` field in trade
- Support load of multiple portfolios
- Save As and Save buttons per portfolio

1.3.5 [1.0.0] 2019-05-03

Added

- Initial release

Python Module Index

m

t

u

Index

H

I

M

N

O

P

R

S

T

U

W