Report on Event-driven Back-testing Engine

An event-driven back-testing engine is a system that automatically receive the market data, generate trading signals and orders for the portfolio based on specific strategy, and execute the orders in live trade through Interactive Brokers API.

There are five important components for the back-testing engine. All of them are designed as abstract base classes (ABC) and free to extend/derive. These classes are Events, DataHandler, Strategy, Portfolio and ExecutionHandler. The workflow for the whole engine is in the same order. Brief introduction on the functionality of each class is as follows.

For Events class, there are four types of events: Market, Signal, Order and Fill Event. These events will be generated in the following classes and handled.

For DataHandler class, it presents an interface for handling both historical or live market data. Derived classes can be distinguished by the formats of data files. There are two abstract methods called get_latest_bar (returns latest N bars in the ticker symbol lists) and update_bars (push latest bars to the symbol structure for all in ticker symbol lists, and return a Market Event object), which will be overwritten in derived classes. In derived class, there are four methods, including the forementioned two. The other two are open_convert_files (store ticker and historical data into dictionary, reindex bars for all by union) and get_new_bar (a generator to get the formatted bar data)

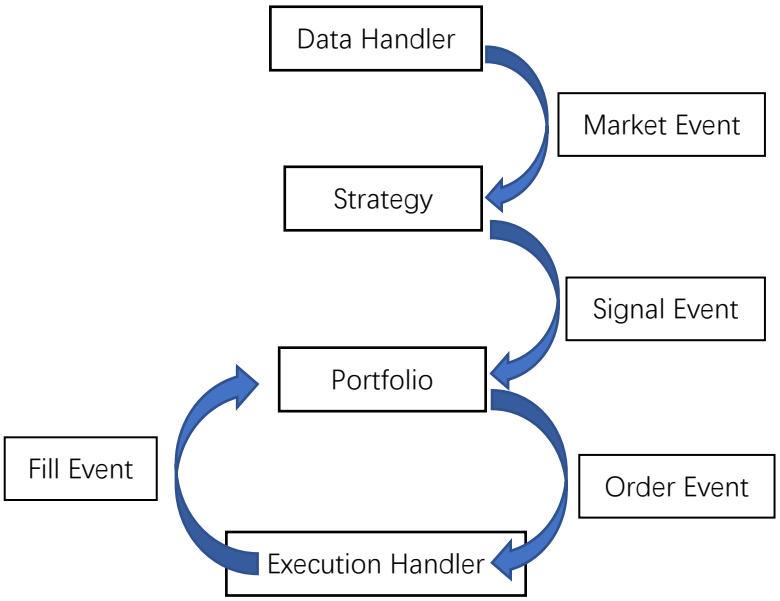
For Strategy class, it presents an interface for taking market data and generating corresponding Signal Events, which are further used by the Portfolio class. This class mainly contain two methods: cal_initial_bought (generate the initial buying status for each ticker in symbol lists) and cal_signals (should receive a market event object to ensure data are updated, generate signals for each ticker in symbol lists based on specific strategy and returns a Signal Event object)

For Portfolio class, it handles the order management associated with current and subsequent positions for a strategy, also carries out risk management across the portfolio. It takes Signal Event object generated from Strategy class and returns Order Event object. There are two abstract methods called update_signal (generate order based on Signal Event object) and update_fill (update the portfolio current positions and holdings from Fill Event object). In derived classes, there are four other methods: construct_all_position (construct positions as 0 for each ticker in symbol lists since start date), construct_all_holdings (construct holdings, cash, commission as 0.0 for each ticker in symbol lists since start date), construct_current_holdings and update_time_index (add a new record to the position matrix for the current market data bar).

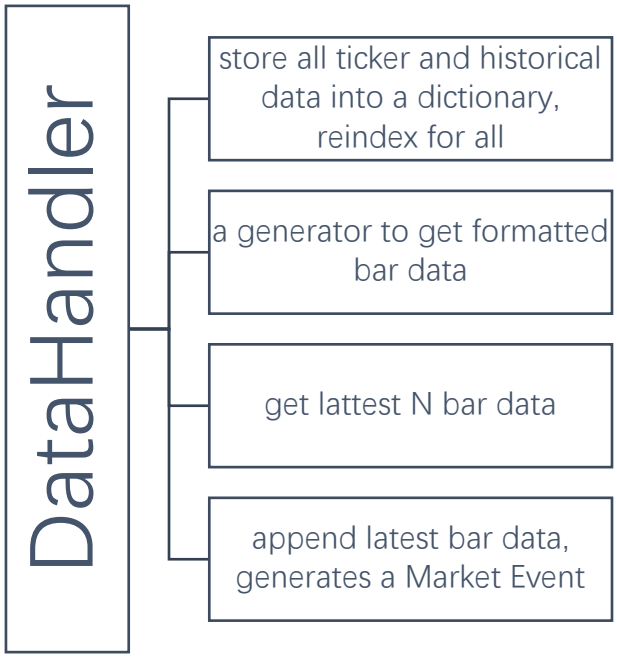
For ExecutionHandler class, it executes the order generated from the Portfolio object by connecting to the live brokerage through API. It will generate a Fill Event object

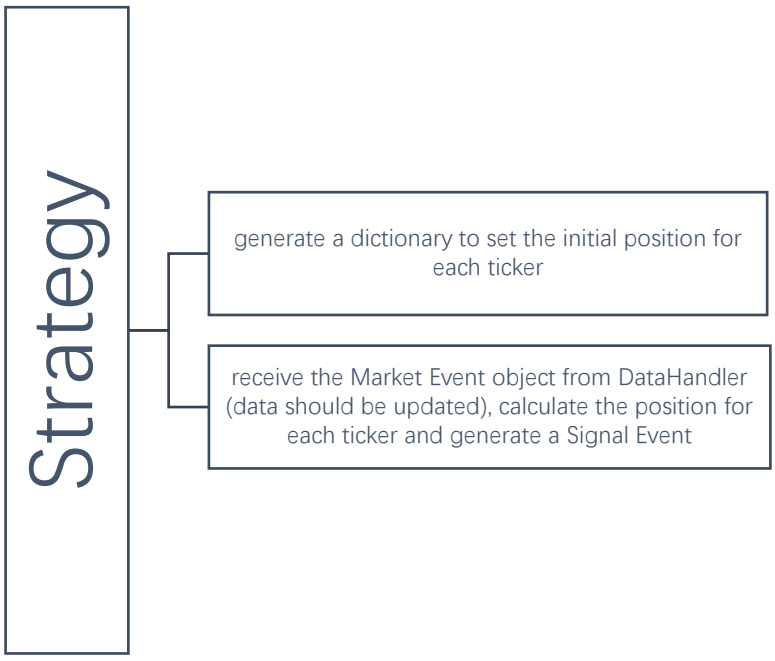
which is in turn used in Portfolio class to update the position and holdings. There are 7 component methods of this class, which can be further split into two categories. Ones are to interact with the Brokerage through API, the others are to generate objects relative to the orders. For the interacting methods, there are reply_handler (handle server reply of Brokerage), connection (connect to the Broker trading platform). For the methods relative to the orders, there are create_contract (create a Contract object defining what will be purchased, at which exchange and in which currency), create_order (create an Order object Market/Limit to go Long/Short), execute_order (execute the order through API, also generate Fill object), create_Fill (create Fill Event object, which will be further used to update the positions and holdings of the portfolio) and create_Fill_dict_entry (container of orderIDs and information from the server, used to avoid duplicate order).

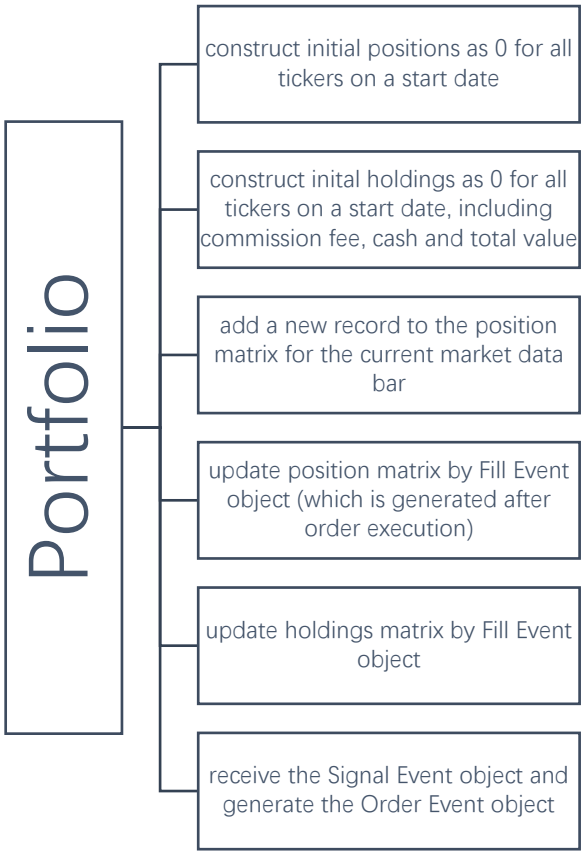
The following chart depicts the general framework of the event-driven engine.

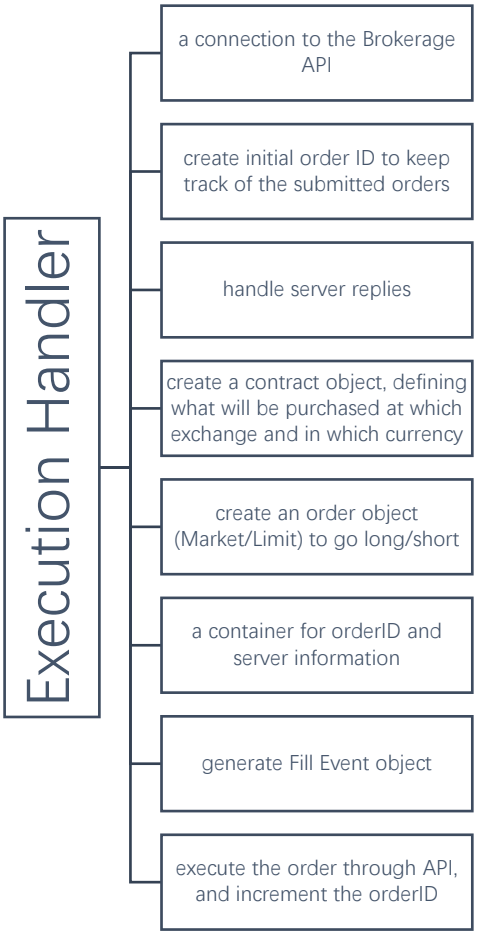


The following chart shows the functionality of each class' methods.


DataHandler
store all ticker and historical data into a dictionary, reindex for all
a generator to get formatted bar data
get lattest N bar data
append latest bar data, generates a Market Event


Strategy
generate a dictionary to set the initial position for each ticker
receive the Market Event object from DataHandler (data should be updated), calculate the position for each ticker and generate a Signal Event


Portfolio
construct initial positions as 0 for all tickers on a start date
construct inital holdings as 0 for all tickers on a start date, including commission fee, cash and total value
add a new record to the position matrix for the current market data bar
update position matrix by Fill Event object (which is generated after order execution)
update holdings matrix by Fill Event object
receive the Signal Event object and generate the Order Event object


Execution Handler
a connection to the Brokerage API
create initial order ID to keep track of the submitted orders
handle server replies
create a contract object, defining what will be purchased at which exchange and in which currency
create an order object (Market/Limit) to go long/short
a container for orderID and server information
generate Fill Event object
execute the order through API, and increment the orderID