Microssa Documentation

Version 1.4

Michael Dinolfo <mike@dinolfo.us>

Contents

1 Overview

Microssa is a free matching engine written in Java which includes a FIX engine and database integration.

1.1 Mission

This software was created with the following goals:

- Free: This software's license (GPL, "copyleft") gives rights to the users while ensuring copies and derivatives remain under such a license. Required dependencies must be under permissive licenses.
- Agnostic: This matching engine does not care what it is trading. It can be things real or virtual, securities, currency, or goods.

However, within the terms the GPL license, you are free to use this software and modify it for any purpose you desire. See the file `LICENSE` in the root project directory for the complete details.

1.2 Examples of Use

This software was created with the following use cases in mind:

- Bitcoin or other cryptocurrency exchange
- Market in a simulated environment, such as a video game
- Virtual stock exchange used to develop a trading algorithm

Currently, Microssa would be a poor choice for:

- Production stock exchange or ECN: this software lacks compliance checks and 3rd party reporting.
- High speed trading: this software is currently not optimized for this purpose.

1.3 Non-Permissive Licenses or Customizations

Contact Michael Dinolfo if you or your organization would like a release of Microssa under a different license, or are interested in the development of custom logic that suits your needs.

1.4 Getting Started

1.4.1 Dependencies

OpenJDK 8 Required to build. Building may also work with Sun's version 8 JDK.

Quickfix/J Required to build. Microssa was last tested with the all jar version 1.6.4 but may work with other configurations. This jar is the primary API called by the FIXInterface class. Quickfix/J has its own dependencies which are required for Microssa to run:

mina tested with version 2.0.17

slf4j api tested with version 1.7.25

slf4j jdk tested with version 1.7.25

make Recommended. Allows the project to be built with relative ease. See the following two subsections for the difference.

expect Recommended. Allows use of the test suite scripts located in the `test` folder.

texlive Optional. Allows rebuilding of this document.

DejaVu Sans font Optional. Allows rebuilding of this document.

LibreOffice Optional. Allows modifying of diagram at the end of this document.

MySQL Optional. A database to read incoming orders and store trade reports.

Connector/J Optional. The API Microssa will use to communicate with a MySQL database.

1.4.2 Building the JAR with make

Go to the home directory of the Microssa project. Run the following to build the project:

```
make
```

This will compile the java files in `src/Microssa`, build a jar file, and place the jar in the `bin` folder.

1.4.3 Building the JAR without make

Go to the home directory of the Microssa project. Run the following to build the project:

```
cd src/Microssa
javac *.java
cd ..
jar cfm Microssa.jar MANIFEST.MF Microssa/
mv Microssa.jar ../bin
```

1.4.4 Configuration

The file `Microssa.cfg` in the `bin` folder contains settings to modify the log file name, ports opened, and matching engine logic. Comments within the file explain each available option and its usage.

1.4.5 Starting Microssa

Change into the `bin` folder and run the shell script `./start-microssa.sh`. Optionally, you can start with `java -jar Microssa.jar`. Starting will not return the prompt, nor anything to it unless there is an exception. Rather, the file `Microssa.log` shows the matching engine's actions. See the `Socket Interfaces` section of this document on how to interact with the matching engine.

1.5 Financial Terminology

The source code and documentation of this project contain a lot of financial language. This section is intended to define and clarify many of those terms.

1.5.1 Order

An order is a party's willingness to buy or sell some amount of a real or virtual object at a specified price. Other attributes of an order serve as identifiers or special instructions. See the section on the `Order Object` for more details.

1.5.2 Symbol

Text identifier for the object that the order is trying to buy or sell. Examples include specific stocks (AAPL for Apple), options, currency (USD v GBP), bonds, or Bitcoins (BTC/BTX).

1.5.3 Match, Execution, Fill

Two orders can match when:

- One is buying.
- One is selling.
- They are for the same Symbol.
- Their prices are compatible: either they have the same price, or the buyer is willing to bid more than what the seller is offering.

Each order then gets and execution, or fill. An order is said to be fully filled or completed if the match, or the sum of all of its matches, is equal to its entire Quantity.

Other conditions can affect match eligibility such as currency, settlement date, and minimum fill quantity.

1.5.4 Time in Force

Specifies the lifetime of an order. The two most common options are:

- Immediate or Cancel: IOC orders enter a matching engine and attempt to execute immediately. These orders then expire, if not fully filled.
- Good for Day: DAY/GFD orders enter a matching engine and attempt to execute immediately. These orders then become passive orders if not fully filled. They can be filled against orders that arrive at a later time. DAY/GFD orders expire at the end of the trading day.

1.5.5 Book, Passive, Aggressive

A book is a list of passive, or resting, orders. These orders had insufficient matches to fully fill them at the time they entered the book. An aggressive order is a new/amended order that enters a matching engine and matches before being written to the book. A match always consists of one passive order and one aggressive order. An IOC order can only be an aggressive order, but a DAY/GFD order can fill either role.

1.5.6 Market, Exchange, ECN

Locations where orders are sent to be matched. Each typically consists of a matching engine, interfaces such as FIX engines or message queues, market data feeds, and execution feeds.

1.5.7 Matching Engine

The part of a market, exchange, or ECN that matches the orders. Microssa is a matching engine.

1.5.8 Interface, FIX Engine, Message Queue

These connections electronically link the matching engine to the outside world. They allow parties to send and manage orders to a matching engine, and to for the engine to notify the parties on the state of their orders.

1.5.9 Market Data

The display of prices and quantities for passive orders. This information is used by traders help them determine what kind of order they want to send to the market.

1.5.10 Execution Feed or Ticker

The display of trades that occurred on the matching engine. Traditionally shows the object traded, the price, and the quantity, but one or both parties may not be shown to protect their anonymity.

1.5.11 Dark Pool

A type of market that does not publish market data about their passive orders. Because a trader has to send an order blindly, and cannot see the current state, it is a ”dark” market.

1.5.12 Price Improvement

When a buyer is willing to bid more than what the seller is offering. For example, someone is willing to sell a Bitcoin for one dollar and someone else is willing to buy at two dollars. There are many schemes on how to resolve the prices these orders get on their fill - here are a couple of examples:

- Midpoint: Match at the mean of the two prices. Microssa uses this scheme. In our Bitcoin example, the match would be at one dollar and fifty cents. Both orders get a fifty cent discount.
- Favor Passive: Match at the aggressive order’s price. If the seller was passive in our Bitcoin example, the match would be at two dollars. The buyer gets no discount and the seller gets a dollar discount.

2 Order Workflow

Below are the descriptions of the high-level steps for the three types of inbound orders messages to the matching engine.

2.1 Order Object

The `Order` object sanity checks the inbound order to make sure there isn't questionable data such as negative prices or blank order IDs. The following fields are a part of an `Order` object:

OrderID: The identifier of the Order. Should be unique.

Symbol: The label/code for the real or virtual good.

Customer: The customer's identifier.

Source: The location from where this `Order` originated. This is stamped on the order by Microssa.

ArriveDate: The date this `Order` arrived.

TIF: The time in force.

Price: The price the customer would like to buy or sell.

Quantity: The maximum amount of goods to buy or sell.

AvailableQuantity: The remaining quantity on the order. Would be less than Quantity if the order was partially filled, or zero if the order was fully filled.

MinFillQuantity: The minimum amount that the order is allowed to be filled. It can be set to the quantity, meaning the customer does not want partial fills. It is also used to prevent very small fills.

Side: Whether this is a buy or sell `Order`.

Currency: The currency-type of the price. Examples are U.S. Dollar (USD) and British Pound (GBP). The order will only match with another order in the same currency.

2.2 New Order

1. Place the order in an `Order` object, which validates the fields on the order.
2. Check we do not have another `Order` with the same ID.

3. Report that we have a new `Order` to Hooks, OrderSocket, and Logger.
4. Attempt to match the `Order` against previously entered, live DAY `Order`s.
5. Add the order to the book if it is a DAY `Order` and was not fully executed by the previous step.
6. Update the market data via PriceSocket.

2.3 Amend Order

1. Place the order in an `Order` object, which validates the fields on the order.
2. Check we have another `Order` with the same ID.
3. Check we are not making a questionable amendment. Typically that means not changing the customer, order ID, symbol, or side.
4. Report that we are amending the `Order` to Hooks, OrderSocket, and Logger.
5. Attempt to match the new `Order` against previous, live DAY `Order`.
6. Remove the old `Order` from the book.
7. Add the new `Order` to the book if it is a DAY `Order` and was not fully executed by the execution step.
8. Update the market data.

2.4 Cancel Order

1. Place the order in an `Order` object, which validates the fields on the order.
2. Validate we have another `Order` with the same ID.
3. Report that we are canceling the `Order` to Hooks, OrderSocket, and Logger.
4. Remove the `Order` from the book.
5. Update the market data.

3 Socket Interfaces

Socket messages are designed to be in a human-readable CSV format. Currently each socket can maintain a single connection.

3.1 General

The sockets send `CONNECTED` after the connection is established. Sending `END` will result in a reply of `BYE` and the socket getting disconnected. Sending an unhandled message will result in a reply of `UNKNOWN COMMAND`; three of these in a row and the socket will send `BYE` and disconnect.

```
$ telnet localhost 2500
Trying ::1...
Connected to localhost.
Escape character is '^]'.
CONNECTED
rm -rf *
UNKNOWN COMMAND
END
BYE
Connection closed by foreign host.
$
```

Sockets will send a PING message every five seconds after receiving the first command from the client in order to test that the connection has not abruptly ended. The client should ignore these messages.

3.2 OrderSocket

Default port is 2500. The first value of inbound and outbound messages define the message type (NEW/AMEND/CANCEL). The message's fields do not have to be in any particular arrangement. Reject messages will contain the value `RejectText=` explaining the error that was encountered.

3.2.1 New Order Message and Replies

New Order Request

```
NEW,OrderID=TESTA,Symbol=BTC,Customer=MD,
ArriveDate=20151003,TIF=DAY,Price=1.0,Quantity=1000.0,
AvailableQuantity=1000.0,Side=B,Currency=USD,MinFillQuantity=100.0
```

New Order Accept

```
NEW,OrderID=TESTA,Customer=MD,Source=OS,Symbol=BTC,
Side=B,Price=1.0,Quantity=1000.0,AvailableQuantity=1000.0,
TIF=DAY,ArriveDate=20151003,Currency=USD,MinFillQuantity=100.0
```

New Order Reject

```
REJECTNEW,OrderID=TESTA,Customer=MD,Source=OS,
Symbol=BTC,Side=B,Price=1.0,Quantity=1000.0,
AvailableQuantity=1000.0,TIF=DAY,ArriveDate=20151003,Currency=USD,
MinFillQuantity=100.0,RejectText=Order already exists in book
```

3.2.2 Amend Order Message and Replies

Amend Order Request

```
AMEND,OrderID=TESTA,Symbol=BTC,Customer=MD,
ArriveDate=20151003,TIF=DAY,Price=1.0,Quantity=2000.0,
AvailableQuantity=2000.0,Side=B,Currency=USD,MinFillQuantity=100.0
```

Amend Order Accept

```
AMEND,OrderID=TESTA,Customer=MD,Source=OS,Symbol=BTC,
Side=B,Price=1.0,Quantity=2000.0,AvailableQuantity=2000.0,
TIF=DAY,ArriveDate=20151003,Currency=USD,MinFillQuantity=100.0
```

Amend Order Reject

```
REJECTAMEND,OrderID=TESTA,Customer=MD,Source=OS,
Symbol=BTC,Side=S,Price=1.0,Quantity=2000.0,
AvailableQuantity=2000.0,TIF=DAY,ArriveDate=20151003,
Currency=USD,MinFillQuantity=100.0,
RejectText=Cannot amend side for order ID=TESTA
```

3.2.3 Cancel Order Message and Replies

Cancel Order Request

```
CANCEL,OrderID=TESTA,Symbol=BTC,Customer=MD,
ArriveDate=20151003,TIF=DAY,Price=1.0,
Quantity=1000.0,AvailableQuantity=1000.0,Side=B,
Currency=USD,MinFillQuantity=100.0
```

Cancel Order Accept

```
CANCEL,OrderID=TESTA,Customer=MD,Source=OS,Symbol=BTC,
Side=B,Price=5.0,Quantity=2000.0,AvailableQuantity=2000.0,
TIF=DAY,ArriveDate=20151003,Currency=USD,MinFillQuantity=100.0
```

Cancel Order Reject

```
REJECTCANCEL,OrderID=TESTA,Customer=MD,Source=OS,
Symbol=BTC,Side=B,Price=1.0,Quantity=1000.0,
AvailableQuantity=1000.0,TIF=DAY,ArriveDate=20151003,
Currency=USD,MinFillQuantity=100.0,
RejectText=Cannot cancel unknown order
```

3.2.4 Other Messages

Match Notification

```
MATCH,OrderID=TESTA,TradePrice=1.0,TradeQuantity=1000.0
```

Completed (Fully Filled) Order

```
COMPLETED,OrderID=TESTA,Customer=MD,Source=OS,
Symbol=BTC,Side=B,Price=1.0,Quantity=2000.0,
AvailableQuantity=0.0,TIF=DAY,ArriveDate=20151003,
Currency=USD,MinFillQuantity=100.0
```

Expired Order

```
EXPIRED,OrderID=TESTA,Customer=MD,Source=OS,
Symbol=BTC,Side=S,Price=1.0,Quantity=1000.0,
AvailableQuantity=1000.0,TIF=IOC,ArriveDate=20151003,
Currency=USD,MinFillQuantity=100.0
```

3.3 PriceSocket

Default port is 2501. The first value of inbound and outbound messages define the message type. Subsequent fields on inbound messages contain the Symbols the session is either subscribing or unsubscribing. There are no acknowledgments on successful subscription changes, but the client will receive a message on a failed subscription.

3.3.1 Subscribe and Unsubscribe

Subscribe Request

```
SUB,BTC,BTA
```

Subscribe Reject

```
REJECT,Symbol=BTC,RejectText=Already subscribed
```

Unsubscribe Request

```
UNSUB,BTC,BTT
```

Unsubscribe Reject

```
REJECT,Symbol=BTT,RejectText=Not subscribed
```

3.3.2 Market Data Snapshot

Snapshot

```
SNAPSHOT,BTC,BID,1.01,800.0,0.99,1000.0,OFFER,
1.02,200.0,1.02,700.0,1.05,400.0
```

Snapshot: No resting orders for Symbol

```
SNAPSHOT,BTC
```

Snapshot: No resting bids for Symbol

```
SNAPSHOT,BTC,OFFER,1.02,200.0,1.02,700.0,1.05,400.0
```

Snapshot: No resting offers for Symbol

```
SNAPSHOT,BTC,BID,1.01,800.0,0.99,1000.0
```

4 Database Interface

4.1 Overview

Microssa supports an interface to a MySQL interface for inbound order new, amend, and cancel actions, and to write trades. The `sql` directory contains table definitions and test orders outside of the testing suite.

See `Microssa.cfg` for details on setting up this interface. The Connector/J interface should be downloaded and placed in the `lib` directory.

4.2 Tables

4.2.1 inbound_orders

This table contains new, amend, and cancel messages inserted from another source. By default this tables is scanned every thirty seconds. If records are found, they will be sent to the matching engine and removed from the table.

The column sequence_number automatically increments and keeps track of the sequence of matches. The column Action determines the type of message: 'N' for new, 'A' for amend, and 'C' for cancel.

4.2.2 trades

This table contains trades completed by the matching engine. This table is not purged by Microssa.

The column sequence_number automatically increments and keeps track of the sequence of matches. The Role column contains 'A' for aggressive, 'P' for passive. See the Financial Terminology section for more information.

5 FIX Interface

5.1 Overview

The Financial Information eXchange (FIX) protocol is used for real-time electronic communications of security transactions and markets. The Microssa matching engine implements a simple FIX engine leveraging the QuickFIX/J API.

The FIX engine is able to process new, cancel/replace, and cancel messages inbound, and sends execution messages outbound for: accept new, accept replace, accept cancel, reject with description in tag 58, expire, partial fill, and fully filled.

At this time Microssa does not include automated tests for FIX.

5.2 Configuration

The FIX configuration is found in `fix-session.cfg`. More information on how to configure this file can be found in the QuickFIX/J documentation on their website.

The connection should remain as an acceptor/inbound as it is customary for recipients of order flow.

6 Component Map

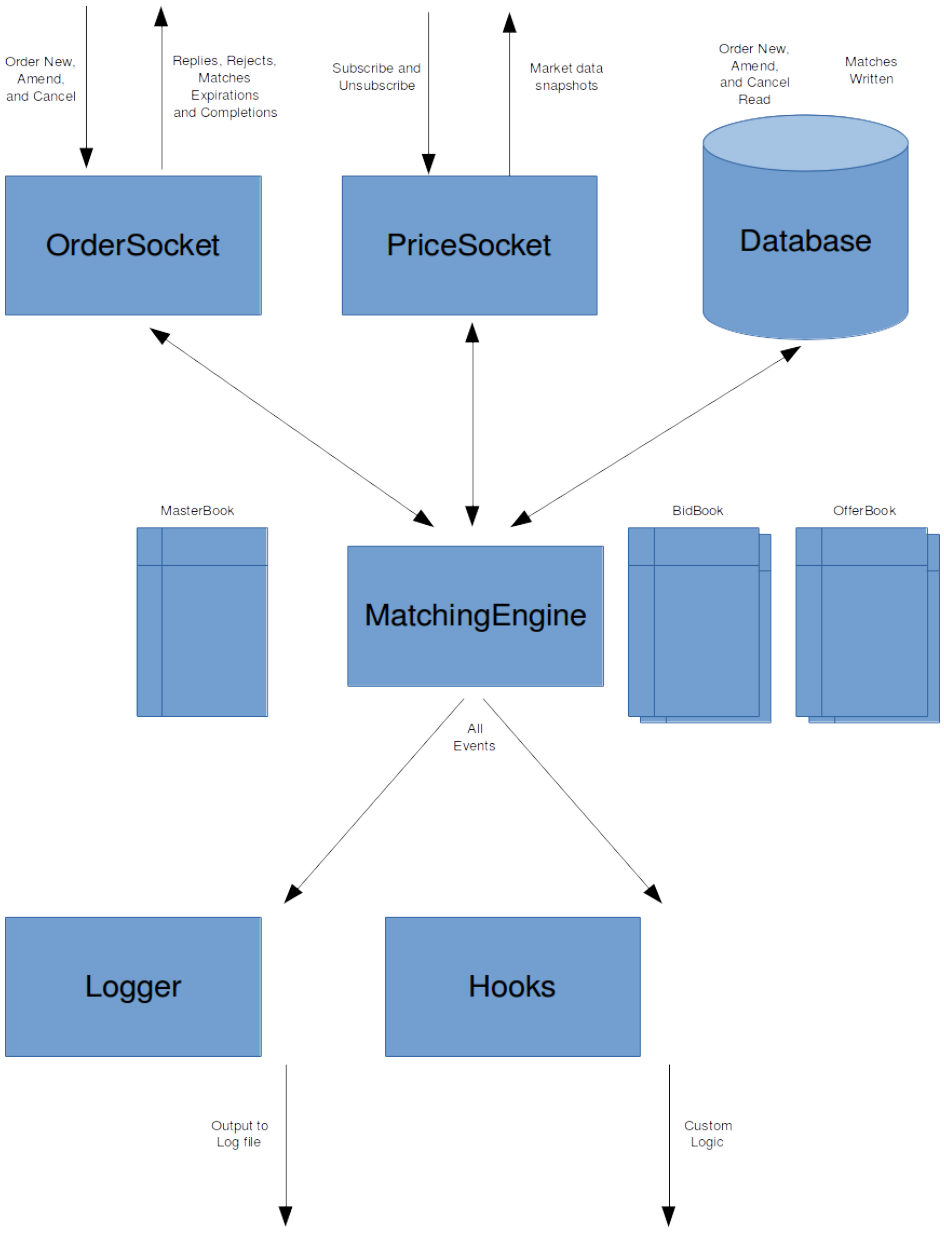

Figure 1: Component Map