Text

Description automatically generated with medium confidence

# QF633 PROJECT REPORT DRAFT

# Code explanation

## csvFeeder.h

1. std::map<std::string, int> column\_pos\_;
   1. we include this line of code under the csvFeeder.h file as a private member variable as it is only relevant to the CsvFeeder class.

## csvFeeder.cpp

1. uint64\_t TimeToUnixMS(std::string ts)
   1. convert a string into Unix timestamp in milliseconds
2. double ConvertToDouble(const std::string &str)
   1. Converts a string to a double. If the string cannot be converted, it returns NaN.
3. bool ReadNextMsg(std::ifstream &file, Msg &msg, std::map<std::string, int> &column\_pos)
   1. Read the next line of data from a CSV file and store it in the Msg structure with tick data.
   2. Store column names and their positions in ‘column\_pos’.
      1. ‘column\_pos’ serves as a dictionary that relates the name of each column to its position in the CSV file.
   3. Also check the csv for CSV for empty lines, timestamps, and if it reaches the end of the file.
4. CsvFeeder::CsvFeeder()
   1. A constructor that takes in a csv file and handles events (FeedListener) based on new data and based on a timer (TimerListener).
5. bool CsvFeeder::Step()
   1. Calls the FeedListener with new data, check the timestamp against the interval, and calls the TimerListener if the interval has crossed. When there are no more messages in the CSV file it returns FALSE.
6. CsvFeeder::~CsvFeeder()
   1. This destructor deallocates resources that were allocated to the CsvFeeder object.

## VolSurfBuilder.h

1. class VolSurfBuilder {}
   1. A template class where the template parameter is ‘Smile’ and is a placeholder for any class that the builder will use.
   2. The class processes the messages coming in and maintains a map ‘currentSurfaceRaw’ that updates upon processing each message.
2. std::map<std::string, TickData> currentSurfaceRaw
   1. A map where the key is the contract name and the value is the tick data.
3. void VolSurfBuilder<Smile>::Process(const Msg& msg)
   1. ‘void Process’ processes the incoming messages and depending on if the message is a snapshot, the current snapshot is discarded to construct a new one from the incoming message. If the incoming is an update, the current market snapshot is updated.
4. void VolSurfBuilder<Smile>::PrintInfo()
   1. Iterates over the map in ‘currentSurfaceRaw’ and prints out the contract name and the implied volatility.
5. std::map<datetime\_t, std::pair<Smile, double> > VolSurfBuilder<Smile>::FitSmiles()
   1. Group tickers in snapshot by expiry, create an instance of ‘Smile’ for each expiry using ‘FitSmile.’
   2. Calculate fitting error and returns a map containing the expiry date and the fitting error.
6. datetime\_t expiryDate = GetExpiry(it->second.UnderlyingIndex)
   1. A helper function to extract the expiry date.

# A screenshot of a computer Description automatically generatedStep 1

Column header read in correctly.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generatedEnsure that the tick data of the same time frame is read in together within the same step. As seen from the log after we detected the change in timestamp, we will reset the file position to prepare for the ReadNextMsg method to process the same line again in the next step. Seen from above figure at row **541** where we ended the snap timestamp, and then we proceeded to read in the **BTC-27MAY22-44000-P** tick data.

A screenshot of a computer

Description automatically generated

Before we process the line, we can see that the position is still the same as previous figure “**97805**” and after reading line it became “**97982**”. Then we proceed to add the tick data.

A screenshot of a computer

Description automatically generated

After reading the **BTC-27MAY22-44000-P** contract we see that the timestamp changed again to **1652572801712** so we reset the position again and read the **BTC-3JUN22-35000-C** in the next step.

A screenshot of a computer

Description automatically generated

On the left is the tick data for **BTC-3JUN22-35000-C** contract. The process then repeats again, reading the same timestamp together in one step.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated We also validated the end of the file to ensure that we will not miss the last contract’s tick data. As seen in the subset data we cut out. Seen on the left **BTC-27MAY22-30000-P** contract is being read in as well.

# Step 2

A screen shot of a computer program

Description automatically generated with low confidence

In the second step we clear the currentSurfaceRaw at line 32 when the Msg is a snapshot and then proceed to populate the currentSurfaceRaw with the tick data. And when it is just a snapshot, we will proceed to replace the currentSurfaceRaw with the updates directly.

A screen shot of a computer program

Description automatically generated with low confidence

In step 2 we also print out the timestamp, contract name and mark IV of the contract.

A screenshot of a computer

Description automatically generated

# Step 3

1. ~~Group market tick data by expiry date~~
2. ~~Pass the data for each expiry date to fitsmile~~
3. Fit the model to market data get fitting error
   1. take the average of BestBidIV and BestAskIV for fitting this is for the ???
   2. Start with equal weight then refine later?(can probably weight by the open interest against all the open interest in the timestamp. KIV)
   3. Describe clearly the formula for the weights in the report

A picture containing text, font, screenshot, number

Description automatically generated

1. Complete the time\_listener in step3.cpp that streams the vol marks and fitting error to the output file   
   TIME,EXPIRY,FUT\_PRICE,ATM,BF25,RR25,BF10,RR10 2022-04-18T00:00:00.139Z,29-APR-2022,38667.63,0.6,0.02,0.05,0.0335,0.067
2. Implement FitSmile in CubicSmile.h to return a CubicSmile that fits the tick data of a given expiry date.

# Smile Fitting Error