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.

# Smile Fitting Error