**API Specification**

The API is fully decoupled from the rest of the application and does not communicate directly with the VIEW (GUI) but rather through the controller package and interfaces. The API consists of two main parts. A FolioModel and a StockModel, along with interfaces for each.

The StockModel contains the ability to buy/sell shares as well as refresh the current values. It also keeps track of key data such as initial buy price and number of shares.

The FolioModel contains an ArrayList of StockModel interfaces. This provides the means to see all stocks associated with that folio as well as interact with each. The rest of the program accesses the stocks through this class and therefore FolioModel provides all the method based functionality that StockModel does. This includes buying/selling shares and refreshing values. It provides some extra functionality in the form of array management and sorting for all the stocks associated with this folio. This is also the level that we provide save and load functionality.

Onto the API itself

***FolioModel***

**String getName():** *Returns the name of the folio*

**void updateValue():** *Updates the total value of the folio. This iterates through the associated stocks and adds their values to total.*

**Double getValue():** *Returns the value of the folio as a Double for use in a table.*

**synchronized void refreshStocks()**: *This method refreshes every stock associated with the current folio. The associated stocks are kept in an ArrayList. As we have manual and automatic refreshes (running on a different thread), as well as forced refreshes on adding and changing stocks, this method is synchronized. This prevents race conditions and deadlocks.*

**ArrayList<iStockModel>:** *Returns the array of stocks as stock interfaces*

**void setStocks(ArrayList<iStockModel> newStocks):** *Sets the array of stocks to the given array, will not refresh any values.*

**ArrayList<StockModel> sort(int sortCode, boolean ascending)**: *sorts the list of stocks. sortCode 0 = sort by ticker, 1 = by name, 2 = by num shares, 3 = price per share, 4 = by value. Returns sorted list*.

**iStockModel newStock(String ticker, String name, int shares)**: *creates new StockModel instance with passed in values and adds it to list of stocks. Returns the new stock if the given arguments are valid. Returns null if not.*

**iStockModel buyStock(iStockModel s, int shares)**: *Buys ‘shares’ amount of shares in the stock ‘s’, returns the updated stock if successful or null if unsuccessful.*

**boolean deleteStock(String ticker)**: *removes the stock with the matching ticker symbol from the ArrayList of associated stocks. Returns true if successful, false if failed.*

**boolean save(String path)**: *Saves this folio and all its stock associated to it to the specified path*, returns true if successful and false if failed.

**static FolioModel load(String path)**: *Loads back in a FolioModel from the specified path. Returns the FolioModel if successful and null if unsuccessful. This method is static as it needs to be accessible without instantiating a FolioModel just to call it.*

**int getId():** *Returns the ID of the current folio*

***StockModel***

There are basic getter methods for every piece of data that is of pertinence to the GUI. These include: double getValue(), double getInitValue(), double getInitBuyPrice(), double getLastKnownPrice(), int getInitialNoOfShares(), double getLow(), double getHigh(), int getNumShares(), String getName(), String getTickerSymbol().

**void buyShares(int amount)**: *increments stored number of shares with passed in amount. This method also updates stock value.*

**boolean sellShares(int amount)**: *Sells ‘amount’ number of shares. If ‘amount’ is more than the current number of shares in the stock, this returns false without doing anything. If successful this will return true.*

**double getGain()**: *Returns the profit made off this stock calculated by: (Current value of stocks)  - (Stored initial buy price \* stored initial number of shares) – (total value of bought stocks since initial purchase) + (total value of sold stocks since purchase)*

**double getTrend()**: *returns a sum of: (The most recently known price -  the stored initial buy price) \* the initial number of shares*

**iStockModel refresh()**: *sends a request to tickertech.com via the StrathQuoteServer to retrieve an updated price for this stock. Updates the stored last known price. Updates the value of this stock. Calls setHigh(double x) and setLow(double x) passing in this new updated price.*

**void setHigh(double lastKnownPrice)**: *Checks to see if the past in last known price is greater than the stored recorded highest price. If it is it will update the stored recorded highest price with this one.*

**void setLow(double lastKnownPrice)**: *Checks to see if the past in last known price is less than the stored recorded lowest price. If it is it will update the stored recorded lowest price with this one.*