# NASDAQ ITCH50 Book Constructor

Generated by Doxygen 1.8.15

# Chapter 1

# Class Index

## 1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

# Chapter 2

# File Index

## 2.1 File List

Here is a list of all documented files with brief descriptions:

# Chapter 3

# Class Documentation

## 3.1 BookConstructor Class Reference

### Public Member Functions

- BookConstructor (const std::string &inputMessageCSV, const std::string &outputMessageCSV, const std::string &outputBookCSV, const std::string &_stock, const size_t &_levels)
- ~BookConstructor ()
- void start (void)
- void next (void)
- bool updateMessage (void)
- void updateBook (void)
- void updatePool (void)
- void WriteBookAndMessage (void)

### 3.1.1 Constructor & Destructor Documentation

#### 3.1.1.1 BookConstructor()

```
BookConstructor::BookConstructor (
            const std::string & inputMessageCSV,
            const std::string & outputMessageCSV,
            const std::string & outputBookCSV,
            const std::string & _stock,
            const size_t & _levels )
```

Class Initializer.

Principal class for the reconstruction of the order book. The constructor also writes the headers to the output files.

**Parameters**

| | | |
|---|---|---|
| in | *inputMessageCSV* | decompressed binary ITCH50 file to read from. |
| in | *_stock* | selected stock. |
| in | *_levels* | selected number of levels for order book. |
| out | *outputBookCSV,outputMessageCSV* | destination files to write order book and stream message. |

#### 3.1.1.2 ~BookConstructor()

```
BookConstructor::~BookConstructor ( )
```

Class deconstructor.

For debug purposes print to std output orders still present at closure. There shouldn't be any.

### 3.1.2 Member Function Documentation

#### 3.1.2.1 next()

```
void BookConstructor::next (
            void  )
```

Process next message. Retain only message affecting the OrderBook (type A,P,D,R,E,C). Reads the message from the Reader interface then if necessary, complete message information retriving information from OrderPool, then updates the OrderBook and OrderPool according to the type of message recived. At the end the Writer writes the book and message (enriched with all additional information) to the two output files.

#### 3.1.2.2 start()

```
void BookConstructor::start (
            void  )
```

Start Book reconstruction.

calls iteratevely the next method untile the Reader has completed the reading.

#### 3.1.2.3 updateBook()

```
void BookConstructor::updateBook (
            void  )
```

Update OrderBook with the current message.

Updates the OrderBook double map accordingly to the type of the message. A: Add the Order to the pool. If key in the map (price) is already there just add the size. Otherwise add the key with corresponding size. R: Replace existing order in the pool, hence cancel completely the existing size and create a new one.

#### 3.1.2.4 updateMessage()

```
bool BookConstructor::updateMessage (
            void  )
```

Complete message information with missing field.

Once a message is readed by the reader this metod retrives missing informations from the order pool, this behaviour depends on the type of the message. Example : Execution messages miss Price -> retrieve order price from the OP through order ID.

A,P: all the informations are already present, stop. D: MPID,size and price information have to be retrived from the Pool. R: MPID,oldSize and oldPrice information have to be retrived from the Pool. E: MPID,size and price have to be retrived from the Pool. C: MPID,size and original price have to be retrived from the Order Pool.

#### 3.1.2.5 updatePool()

```
void BookConstructor::updatePool (
            void  )
```

Update OrderPool with the current Message.

Using the message attribute in the BookConstructor class to updates the pool.

- A: Add order to OrderPool.
- R: Delete order and add new one.
- D: Delete (partially or totally) order.
- E: Execute (partially or totally) order.
- C: Execute order at different price.
- P: Execute hidden order. Does not affect the book.

#### 3.1.2.6 WriteBookAndMessage()

```
void BookConstructor::WriteBookAndMessage (
            void  )
```

Write in output OrderBook state and message stream through Writer class.

The documentation for this class was generated from the following files:

- include/BookConstructor.hpp
- src/BookConstructor.cpp

## 3.2 Message Class Reference

### Public Member Functions

- **Message** (const char &type, const id_type &id, const time_type &timestamp)
- void setType (const char &)
- void **setId** (const id_type &)
- void **setTimeStamp** (const time_type &)
- void **setSide** (const side_type &)
- void **setPrice** (const price_type &)
- void **setRemSize** (const size_type &)
- void **setCancSize** (const size_type &)
- void **setExecSize** (const size_type &)
- void **setOldId** (const id_type &id)
- void **setOldPrice** (const price_type &)
- void **setOldSize** (const size_type &)
- void **setMPID** (const char &)
- char **getType** (void) const
- id_type **getId** (void) const
- time_type **getTimeStamp** (void) const
- side_type **getSide** (void) const
- price_type **getPrice** (void) const
- size_type **getRemSize** (void) const
- size_type **getCancSize** (void) const
- size_type **getExecSize** (void) const
- id_type **getOldId** (void) const
- price_type **getOldPrice** (void) const
- size_type **getOldSize** (void) const
- const char * **getMPID** (void) const
- bool **isEmpty** (void) const
- std::string getString (void) const
- void **print** (void) const

### 3.2.1 Member Function Documentation

#### 3.2.1.1 getString()

```
std::string Message::getString (
            void  ) const
```

Get string representation for writing into the csv

**Returns**

string representation of message. If field is not being setted it is just an empty char separated by commas.

#### 3.2.1.2 setType()

```
void Message::setType (
            const char & _type )
```

Setter for the message. Transforms the Nasdaq type definitions in ours.

- NASDAQ --> Custom
- A,F --> (A)dd
- D,X --> (D)elete
- U --> (R)eplace
- E --> (E)xecution
- P --> P, hidden execution
- C --> C, execution at different price

**Parameters**

| | | |
|---|---|---|
| in | *_type* | type string: according to the definition of NASDAQ |

The documentation for this class was generated from the following files:

- include/Message.hpp
- src/Message.cpp

## 3.3 Order Class Reference

### Public Member Functions

- **Order** (id_type _id, side_type _side, size_type _size, price_type _price, const char *_mpid)
- void addSize (size_type size)
- id_type **getId** (void) const
- side_type **getSide** (void) const
- size_type **getSize** (void) const
- price_type **getPrice** (void) const
- const char * **getMPID** (void) const
- void **print** (void) const
- bool isEmpty (void) const

### 3.3.1 Member Function Documentation

#### 3.3.1.1 addSize()

```
void Order::addSize (
            size_type size )
```

Add or subtract size to the order.

**Parameters**

| | | |
|---|---|---|
| in | *_size* | : size to add or detract (if size is negative) to the order |

#### 3.3.1.2 isEmpty()

```
bool Order::isEmpty (
            void  ) const
```

Check whether the Order is unsetted or not.

**Returns**

bool, 1 is unsetted (Empty), 0 if setted.

The documentation for this class was generated from the following files:

- include/Order.hpp
- src/Order.cpp

## 3.4 OrderBook Class Reference

```
#include <OrderBook.hpp>
```

### Public Member Functions

- std::string getString (const size_t &) const
- void modifySize (price_type, size_type, side_type)
- void **setTimeStamp** (const time_type &)
- bool checkBookConsistency (void)

### 3.4.1 Detailed Description

Class containing list of buy and sell orders for a specific security organized by price level into 2 ordered maps. An order book lists the number of shares being bid or offered at each price point available, keeping track of time of every change made.

### 3.4.2 Member Function Documentation

#### 3.4.2.1 checkBookConsistency()

```
bool OrderBook::checkBookConsistency (
            void  )
```

Check if the biggest bid price is less than smallest ask

**Returns**

bool value of the check. 1 OK, 0 KO.

#### 3.4.2.2 getString()

```
std::string OrderBook::getString (
            const size_t & level ) const
```

Make comma-separated string from information available in the OrderBook about the best bid/ask prices and corresponding sizes up to number of levels : "1.BidPrice, 1.BidSize,1.AskPrice,1.AskSize,..,level.BidPrice, level.Bid↩Size,level.AskPrice,level.AskSize"

**Parameters**

| | | |
|---|---|---|
| in | *level* | up to what level to write the price/size tuple. |

#### 3.4.2.3 modifySize()

```
void OrderBook::modifySize (
            price_type price,
            size_type size,
            side_type side )
```

Performs actions on the double map representing the OrderBook

**Parameters**

| | | |
|---|---|---|
| in | *price* | modify map corresponding to price |
| in | *size* | add (or delete if size is negative) the size corresponding to price |
| in | *side* | 0 for buy side and 1 for sell side. |

The documentation for this class was generated from the following files:

- include/OrderBook.hpp
- src/OrderBook.cpp

## 3.5 OrderPool Class Reference

`#include <OrderPool.hpp>`

### Public Member Functions

- Order searchOrderPool (id_type)
- void addToOrderPool (id_type, bool, size_type, price_type, const char *)
- void modifyOrder (id_type, size_type)
- bool isEmpty (void) const
- void printIds (void) const

### 3.5.1 Detailed Description

The class tracks all Order objects created. When an “A” (or “F”) message comes in, it creates a Order object in the OrderPool. When subsequently a message comes in indicating limit order cancellation (“X” and “D”) or a limit order execution (“E”), the information about the price and size of the original limit order is retrieved from the OrderPool using common order ID.

### 3.5.2 Member Function Documentation

#### 3.5.2.1 addToOrderPool()

```
void OrderPool::addToOrderPool (
            id_type idOrder,
            bool side,
            size_type size,
            price_type price,
            const char * mpid )
```

Initialize and add an Order to the OrderPool

**Parameters**

| | | |
|---|---|---|
| in | *idOrder* | id of the order to add |
| in | *side* | side of the order to add (0 for buy and 1 for sell) |
| in | *size* | size of the order to add to the pool |
| in | *price* | limit price of the order to add |

#### 3.5.2.2 isEmpty()

```
bool OrderPool::isEmpty (
            void  ) const
```

Check whether the OrderPool map is empty

**Returns**

book, 1 if empty, 0 if not.

#### 3.5.2.3 modifyOrder()

```
void OrderPool::modifyOrder (
            id_type idOrder,
            size_type size = 0 )
```

Delete size of an order in the OrderPool.

If the remaining size is zero then order is deleted from the OrderPool. size is always subtracted from the order.

**Parameters**

| | | |
|---|---|---|
| in | *idOrder* | id of the order to modify |
| in | *size* | size to subtract from the order. |

#### 3.5.2.4 printIds()

```
void OrderPool::printIds (
            void  ) const
```

Prints id of all orders in the OrderPool.

It's used at the end to check if the OrderPool is empty (it should be).

#### 3.5.2.5 searchOrderPool()

```
Order OrderPool::searchOrderPool (
            id_type idOrder )
```

Look for the Order specified by the id in the OrderPool

**Parameters**

| | | |
|---|---|---|
| in | *idOrder* | : id relative to the order quaried |

**Returns**

Order with id equals to idOrder.

The documentation for this class was generated from the following files:

- include/OrderPool.hpp
- src/OrderPool.cpp

## 3.6 Reader Class Reference

### Public Member Functions

- Reader (const std::string &fileName, const std::string &stock)
- Reader (const std::string &_stock)
- bool **isValid** (void) const
- Message createMessage (void)
- bool **eof** (void)
- void printProgress (void)
- virtual void readBytesIntoMessage (const long &)
- virtual void skipBytes (const long &)
- void **setMessage** (const char *)
- virtual char **getKey** (void)
- std::string **getFileName** (void) const
- std::string **getStock** (void) const

### 3.6.1 Constructor & Destructor Documentation

#### 3.6.1.1 Reader() [1/2]

```
Reader::Reader (
            const std::string & fileName,
            const std::string & stock )
```

Constructor for Reader class

If unable to open file to read print to standard error a Message. If file has been opened correctly, write it to standard optput.

**Parameters**

| | | |
|---|---|---|
| in | *_fileName* | destination csv files to update. |
| in | *_stock* | For performace reasons, the Reader class will discard directly all messages clearly related to other stocks |

#### 3.6.1.2 Reader() [2/2]

```
Reader::Reader (
```

```
            const std::string & _stock )
```

Alternative Constructor for Reader class

Constructor used in tests where we do not need a fileName.

**Parameters**

| | | |
|---|---|---|
| in | *_stock* | For performace reasons, the Reader class will discard directly all messages clearly related to other stocks |

### 3.6.2 Member Function Documentation

#### 3.6.2.1 createMessage()

```
Message Reader::createMessage (
            void  )
```

Reads bytes from the stream and create a message

Main function of the class. Creates a Message object from the file stream and return a message to the BookConstructor class.

**Returns**

Message created from the read bytes.

**Warning**

Dead code is still present in the method. Might be used to parse the entire input ITCH50 file for debug purposes

#### 3.6.2.2 printProgress()

```
void Reader::printProgress (
            void  )
```

Progress updates

Writes to standard output a progress message with the number of messages analyzed up to now and average number of messages per second since the beginnning.

#### 3.6.2.3 readBytesIntoMessage()

```
void Reader::readBytesIntoMessage (
            const long & size )  [virtual]
```

Reads n bytes from the opend file

Reads from the file into the message c-string attribute of the Reader class the specified number of bytes.

@params[in] size Number of bytes to read from the stream.

#### 3.6.2.4 skipBytes()

```
void Reader::skipBytes (
            const long & size )  [virtual]
```

Skips n bytes from the stream

Discard from the file the specified number of bytes. Used mainly in the tests.

@params[in] size Number of bytes to discard from the stream.

The documentation for this class was generated from the following files:

- include/Reader.hpp
- src/Reader.cpp

## 3.7 Writer Class Reference

### Public Member Functions

- Writer (const std::string &fileName)
- void writeLine (const std::string &)
- std::string **getFileName** (void) const

### 3.7.1 Constructor & Destructor Documentation

#### 3.7.1.1 Writer()

```
Writer::Writer (
            const std::string & fileName )
```

Constructor for Writer class

If unable to open file to read print to standard error a message. If file has been opened correctly, write it to standard optput.

**Parameters**

| | | |
|---|---|---|
| in | *_fileName* | destination csv files to update. |

### 3.7.2 Member Function Documentation

#### 3.7.2.1 writeLine()

```
void Writer::writeLine (
            const std::string & stringToWrite )
```

Writes string to stream

It used to write the Message and the OrderBook strings to the outfiles.

**Parameters**

| | | |
|---|---|---|
| in | *stringToWrite* | string to write to the csv. |

The documentation for this class was generated from the following files:

- include/Writer.hpp
- src/Writer.cpp

# Chapter 4

# File Documentation

## 4.1 src/main.cpp File Reference

```
#include <iostream>
#include <string>
#include <BookConstructor.hpp>
```

### Functions

- int main (int argc, char ∗argv[ ])

### 4.1.1 Detailed Description

Declaration of main function.

### 4.1.2 Function Documentation

#### 4.1.2.1 main()

```
int main (
            int argc,
            char * argv[] )
```

main

Interface to use the BookRecustruction Class WARNING: this is lightweight and do not perform checks for file existence, good formatting of inputs ecc..

**Parameters**

| | | |
|---|---|---|
| in | *argc* | An integer argument count of the command line arguments |
| in | *argv* | An argument vector of the command line arguments: |
| in | *path* | to the unzipped ITCH raw data binary file |
| in | *path* | to the directory to which the output book csv file will be written to (add trailing backslash!) |
| in | *path* | to the directory to which the output message csv file will be written to (add trailing backslahs!) |
| in | *integer* | this is the depth of the orderbook, for bid and ask side |
| in | *name* | string of the stock you want the reconstruct the book |

## 4.2 src/utility.cpp File Reference

```
#include <utility.hpp>
```

### Functions

- std::string getFileName (const std::string &path)
- uint16_t bswap_16 (uint16_t value)
- uint32_t bswap_32 (uint32_t value)
- uint64_t bswap_64 (uint64_t value)
- uint16_t parse_uint16 (char ∗a)
- uint32_t parse_uint32 (char ∗a)
- uint64_t parse_uint64 (char ∗a)
- uint64_t parse_ts (char ∗a)

### Variables

- side_type **SIDE_DEFAULT** = 0
- id_type **ID_DEFAULT** = LLONG_MAX
- price_type **PRICE_DEFAULT** = -1
- size_type **SIZE_DEFAULT** = -1

### 4.2.1 Detailed Description

Declaration of default values.

### 4.2.2 Function Documentation

#### 4.2.2.1 bswap_16()

```
uint16_t bswap_16 (
            uint16_t value )
```

Utility function for swapping 16 bits from little endian to big endian format.

Since the binary file is written in big endian and most Unix systems are little endian, we defined this utility functions to swap endianess. Uses binary masks to perform this operation.

**Parameters**

| | | |
|---|---|---|
| in | *value* | unsigned 16 type corresponding to the 16 bits in big endian to swap into little endian. |

**Returns**

uint16_t value of the swapped number

**Warning**

Assumes that the machine is little endian and hence the swapping is indeed necessary. Otherwise no swapping is needed. This check is not performed.

#### 4.2.2.2 bswap_32()

```
uint32_t bswap_32 (
            uint32_t value )
```

Utility function for swapping 32 bits from little endian to big endian format.

Since the bynary file is written in big endian and most Unix systems are little endian, we defined this utility functions to swap endianess. Uses binary masks to perform this operation.

**Parameters**

| | | |
|---|---|---|
| in | *value* | unsigned 16 type corresponding to the 32 bits in big endian to swap into little endian. |

**Returns**

uint32_t value of the swapped number

**Warning**

Assumes that the machine is little endian and hence the swapping is indeed necessary. Otherwise no swapping is needed. This check is not performed.

#### 4.2.2.3 bswap_64()

```
uint64_t bswap_64 (
            uint64_t value )
```

Utility function for swapping 64 bits from little endian to big endian format.

Since the bynary file is written in big endian and most Unix systems are little endian, we defined this utility functions to swap endianess. Uses binary masks to perform this operation.

**Parameters**

| | | |
|---|---|---|
| in | *value* | unsigned 64 type corresponding to the 64 bits in big endian to swap into little endian. |

**Returns**

uint64_t value of the swapped number

**Warning**

Assumes that the machine is little endian and hence the swapping is indeed necessary. Otherwise no swapping is needed. This check is not performed.

#### 4.2.2.4 getFileName()

```
std::string getFileName (
            const std::string & s )
```

Simple utility function for get the file name from a path string

**Parameters**

| | | |
|---|---|---|
| in | *path* | string of the path of the file. Should also work for the separator "\\" (WINDOWS). |

**Returns**

nameFile string of the file name.

#### 4.2.2.5 parse_ts()

```
uint64_t parse_ts (
            char * a )
```

Utility function for parsing 48 bits data (for time stamp)

This reads from a char array pointer (C-style) 48 bits and return the swapped correpsonding number

**Parameters**

| | | |
|---|---|---|
| in | *a* | char pointer to the 48 bits to parse |

**Returns**

uint64_t number corresponding to the swapped data (48 bits) pointed by the char array

**Warning**

Assumes that the machine is little endian and hence the swapping is indeed necessary. Otherwise no swapping is needed. This check is not performed.

#### 4.2.2.6 parse_uint16()

```
uint16_t parse_uint16 (
            char * a )
```

Utility function for parsing 16 bits data.

This reads from a char array pointer (C-style) 16 bits and return the swapped corresponding number

**Parameters**

| | | |
|---|---|---|
| in | *a* | char pointer to the 16 bits to parse |

**Returns**

uint16_t number corresponding to the swapped data (16 bits) pointed by the char array

**Warning**

Assumes that the machine is little endian and hence the swapping is indeed necessary. Otherwise no swapping is needed. This check is not performed.

#### 4.2.2.7 parse_uint32()

```
uint32_t parse_uint32 (
            char * a )
```

Utility function for parsing 32 bits data.

This reads from a char array pointer (C-style) 32 bits and return the swapped corresponding number

**Parameters**

| | | |
|---|---|---|
| in | *a* | char pointer to the 32 bits to parse |

**Returns**

uint32_t number corresponding to the swapped data (32 bits) pointed by the char array

**Warning**

Assumes that the machine is little endian and hence the swapping is indeed necessary. Otherwise no swapping is needed. This check is not performed.

#### 4.2.2.8 parse_uint64()

```
uint64_t parse_uint64 (
            char * a )
```

Utility function for parsing 64 bits data.

This reads from a char array pointer (C-style) 64 bits and return the swapped corresponding number

**Parameters**

| | | |
|---|---|---|
| in | *a* | char pointer to the 64 bits to parse |

**Returns**

uint64_t number corresponding to the swapped data (64 bits) pointed by the char array

**Warning**

Assumes that the machine is little endian and hence the swapping is indeed necessary. Otherwise no swapping is needed. This check is not performed.

# Index