

ATS

Generated by Doxygen 1.9.6

1 ATS	1
1.1 Prerequisites	1
1.2 Quick Start Overview	1
1.2.1 Cloning	1
1.2.2 Building	1
1.2.3 Example	2
1.3 Documentation	2
2 Namespace Index	3
2.1 Namespace List	3
3 Hierarchical Index	5
3.1 Class Hierarchy	5
4 Class Index	7
4.1 Class List	7
5 File Index	9
5.1 File List	9
6 Namespace Documentation	11
6.1 ats Namespace Reference	11
6.1.1 Detailed Description	12
6.1.2 Enumeration Type Documentation	12
6.1.2.1 OrderType	12
6.1.2.2 Side	13
6.1.3 Function Documentation	13
6.1.3.1 OrderTypeToString()	13
6.1.3.2 SideToString()	13
6.1.3.3 stringToOrderType()	14
6.1.3.4 stringToSide()	14
7 Class Documentation	15
7.1 ats::BinanceExchangeManager Class Reference	15
7.1.1 Detailed Description	16
7.1.2 Constructor & Destructor Documentation	17
7.1.2.1 BinanceExchangeManager()	17
7.1.3 Member Function Documentation	17
7.1.3.1 cancelOrder()	17
7.1.3.2 getOpenOrders()	17
7.1.3.3 getOrderStatus()	18
7.1.3.4 getPrice()	18
7.1.3.5 getTradeHistory()	18
7.1.3.6 getUserInfo()	19

7.1.3.7 isRunning()	19
7.1.3.8 jsonToOrder()	19
7.1.3.9 jsonToTrade()	20
7.1.3.10 modifyOrder()	20
7.1.3.11 sendOrder()	20
7.2 ats::ExchangeManager Class Reference	21
7.2.1 Detailed Description	22
7.2.2 Constructor & Destructor Documentation	22
7.2.2.1 ExchangeManager()	22
7.2.3 Member Function Documentation	22
7.2.3.1 cancelOrder()	22
7.2.3.2 getOpenOrders()	22
7.2.3.3 getOrderStatus()	23
7.2.3.4 getPrice()	23
7.2.3.5 getTradeHistory()	23
7.2.3.6 modifyOrder()	24
7.2.3.7 sendOrder()	24
7.2.4 Member Data Documentation	24
7.2.4.1 mOrderManager	25
7.3 ats::MarketData Class Reference	25
7.3.1 Detailed Description	25
7.3.2 Constructor & Destructor Documentation	26
7.3.2.1 MarketData() [1/2]	26
7.3.2.2 MarketData() [2/2]	27
7.3.3 Member Function Documentation	27
7.3.3.1 getPrice()	27
7.3.3.2 getQtyForPrice()	27
7.3.3.3 isRunning()	28
7.3.3.4 subscribe()	28
7.3.3.5 unsubscribe()	28
7.4 ats::Order Struct Reference	29
7.4.1 Detailed Description	29
7.4.2 Constructor & Destructor Documentation	29
7.4.2.1 Order()	29
7.4.3 Member Data Documentation	30
7.4.3.1 emsId	30
7.4.3.2 icebergQty	30
7.4.3.3 id	30
7.4.3.4 price	30
7.4.3.5 quantity	31
7.4.3.6 recvWindow	31
7.4.3.7 side	31

7.4.3.8 stopPrice	31
7.4.3.9 symbol	31
7.4.3.10 timeInForce	31
7.4.3.11 type	31
7.5 ats::OrderManager Class Reference	32
7.5.1 Detailed Description	32
7.5.2 Constructor & Destructor Documentation	32
7.5.2.1 OrderManager()	32
7.5.2.2 ~OrderManager()	33
7.5.3 Member Function Documentation	33
7.5.3.1 createOrder()	33
7.5.3.2 getOldestOrder()	33
7.5.3.3 hasOrders()	33
7.5.3.4 isRunning()	34
7.5.3.5 processOrder()	34
7.5.3.6 processOrders()	34
7.5.3.7 run()	34
7.5.3.8 start()	34
7.5.3.9 stop()	35
7.6 ats::Position Struct Reference	35
7.6.1 Detailed Description	35
7.6.2 Constructor & Destructor Documentation	35
7.6.2.1 Position()	35
7.6.3 Member Function Documentation	36
7.6.3.1 total()	36
7.7 ats::PositionManager Class Reference	36
7.7.1 Detailed Description	37
7.7.2 Constructor & Destructor Documentation	37
7.7.2.1 PositionManager()	37
7.7.3 Member Function Documentation	37
7.7.3.1 getPnL()	37
7.7.3.2 getPosition()	37
7.7.3.3 isRunning()	38
7.7.3.4 updatePosition()	38
7.8 ats::RiskManager Class Reference	38
7.8.1 Detailed Description	38
7.9 ats::Strategy Class Reference	39
7.9.1 Detailed Description	40
7.9.2 Constructor & Destructor Documentation	40
7.9.2.1 Strategy()	40
7.9.3 Member Function Documentation	40
7.9.3.1 getSignal()	40

7.9.3.2 isRunning()	41
7.10 ats::Trade Class Reference	41
7.10.1 Detailed Description	41
7.10.2 Constructor & Destructor Documentation	41
7.10.2.1 Trade()	41
8 File Documentation	43
8.1 ats.h File Reference	43
8.1.1 Detailed Description	43
8.2 ats.h	44
8.3 BinanceExchangeManager.h File Reference	44
8.3.1 Detailed Description	44
8.4 BinanceExchangeManager.h	45
8.5 ExchangeManager.h File Reference	46
8.5.1 Detailed Description	46
8.6 ExchangeManager.h	46
8.7 MarketData.h File Reference	47
8.7.1 Detailed Description	47
8.8 MarketData.h	48
8.9 OrderManager.h File Reference	48
8.9.1 Detailed Description	49
8.10 OrderManager.h	50
8.11 PositionManager.h File Reference	51
8.11.1 Detailed Description	52
8.12 PositionManager.h	52
8.13 RiskManager.h File Reference	53
8.13.1 Detailed Description	53
8.14 RiskManager.h	53
8.15 Strategy.h File Reference	53
8.15.1 Detailed Description	54
8.16 Strategy.h	54
8.17 Trade.h File Reference	55
8.17.1 Detailed Description	55
8.18 Trade.h	55
Index	57

Chapter 1

ATS

ATS is an open-source implementation of an algorithmic trading system in C++.

It currently supports the Binance Exchange but the project is modular and any exchange can be added.

1.1 Prerequisites

The project requires C++ 14 and CMake.

Submodules include [binance-cxx-api](#), [GoogleTest](#) and their dependencies.

1.2 Quick Start Overview

1.2.1 Cloning

```
git clone --recurse-submodules -j4 https://github.com/anouarac/algo-trading.git
```

1.2.2 Building

```
$ cd algo-trading
$ mkdir build
$ cd build
$ cmake ..
$ make -j4
```

You should now be able to add your own code after importing [ats.h](#).

1.2.3 Example

Working with this library can go as follows with the Binance EMS:

- Place Binance Spot Net API keys in `$HOME/.binance/key` and `$HOME/.binance/secret`, or `$HOME/.binance/test_key` and `$HOME/.binance/test_secret` for the Spot Test Net.

- Import the following libraries

```
#include <iostream>
#include "ats.h"
#include "json/json.h"
#include "binance_logger.h"
```

- Setup Logger

```
binance::Logger::set_debug_level(1);
binance::Logger::set_debug_logfp(stderr);
```

- Initialise OMS and Binance EMS

```
ats::OrderManager oms;
ats::BinanceExchangeManager b_ems(oms, 1);
```

- Interact with EMS

```
Json::Value result;
b_ems.getUserInfo(result);
std::cout << result.toStyledString() << std::endl;
std::cout << b_ems.getPrice("ETHUSDT") << std::endl;
```

1.3 Documentation

For further details check the documentation.

Chapter 2

Namespace Index

2.1 Namespace List

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

ats	Namespace for the algorithmic trading system	11
---------------------	--	--------------------

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

ats::ExchangeManager	21
ats::BinanceExchangeManager	15
ats::MarketData	25
ats::Order	29
ats::OrderManager	32
ats::Position	35
ats::PositionManager	36
ats::RiskManager	38
ats::Strategy	39
ats::Trade	41

Chapter 4

Class Index

4.1 Class List

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

ats::BinanceExchangeManager	Abstract class that defines the interface for managing orders on the Binance exchange	15
ats::ExchangeManager	Abstract class that defines the interface for managing orders on an exchange	21
ats::MarketData	Handles the streaming of market data for the trading system	25
ats::Order	The Order struct represents an order to be placed on an exchange	29
ats::OrderManager	A class for managing orders	32
ats::Position	Represents a position with its quantity and price	35
ats::PositionManager	Manages the positions and the PnL of a trading system	36
ats::RiskManager	A class that handles risk management	38
ats::Strategy	Defines an abstract interface for trading strategies	39
ats::Trade	Class representing a trade executed on an exchange	41

Chapter 5

File Index

5.1 File List

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

ats.h	Main header file for the ATS library	43
BinanceExchangeManager.h	This class is responsible for managing orders and interacting with the Binance exchange API. This class implements the ExchangeManager interface and provides functionality to send, modify, and cancel orders, as well as get the status of open orders, get the trade history, and get the current price of a symbol. It also provides a method to get the user's account information from the Binance API	44
ExchangeManager.h	Contains an abstract class ExchangeManager The ExchangeManager class is an abstract class that defines the interface for managing orders on an exchange	46
MarketData.h	Contains the declaration of the MarketData class, which handles the streaming of market data for the trading system	47
OrderManager.h	Contains the declaration of the OrderManager class and related enums and structs	48
PositionManager.h	This header file contains the declaration of the PositionManager class, which is responsible for managing and tracking the open positions and PnL of a trading strategy. The PositionManager class provides functionality for starting and stopping a separate thread for updating the open positions and PnL, as well as updating the position for a specific symbol	51
RiskManager.h	53
Strategy.h	Defines an abstract interface for trading strategies	53
Trade.h	Contains the Trade class definition	55

Chapter 6

Namespace Documentation

6.1 ats Namespace Reference

Namespace for the algorithmic trading system.

Classes

- class [BinanceExchangeManager](#)
The [BinanceExchangeManager](#) class is an abstract class that defines the interface for managing orders on the Binance exchange.
- class [ExchangeManager](#)
The [ExchangeManager](#) class is an abstract class that defines the interface for managing orders on an exchange.
- class [MarketData](#)
Handles the streaming of market data for the trading system.
- struct [Order](#)
The [Order](#) struct represents an order to be placed on an exchange.
- class [OrderManager](#)
A class for managing orders.
- struct [Position](#)
Represents a position with its quantity and price.
- class [PositionManager](#)
Manages the positions and the PnL of a trading system.
- class [RiskManager](#)
A class that handles risk management.
- class [Strategy](#)
Defines an abstract interface for trading strategies.
- class [Trade](#)
Class representing a trade executed on an exchange.

Enumerations

- enum [OrderType](#) {
 [LIMIT](#) , [MARKET](#) , [STOP_LOSS](#) , [STOP_LOSS_LIMIT](#) ,
 [TAKE_PROFIT](#) , [TAKE_PROFIT_LIMIT](#) , [LIMIT_MAKER](#) , [OTCOUNT](#) }
Enum for different types of orders.
- enum [Side](#) { [BUY](#) , [SELL](#) , [SCOUNT](#) }
Enum for buy/sell side of an order.

Functions

- `std::string OrderTypeToString (OrderType t)`
Converts OrderType enum value to string.
- `OrderType stringToOrderType (const std::string &s)`
Converts a string to an OrderType enum value.
- `std::string SideToString (Side t)`
Converts Side enum value to string.
- `Side stringToSide (const std::string &s)`
Converts a string to a Side enum value.

6.1.1 Detailed Description

Namespace for the algorithmic trading system.

This namespace contains all the classes and functions for the algorithmic trading system. It includes classes for managing orders, trades, accounts, and exchanges, as well as functions for analyzing market data, making trading decisions, and executing trades.

The system is designed to be extensible, with different types of exchanges and strategies able to be added through inheritance and polymorphism. The core functionality of the system is provided by the [ExchangeManager](#) and [StrategyManager](#) classes, which coordinate the interactions between exchanges, accounts, orders, and trades.

The system uses the JSONCPP library for parsing and manipulating JSON data, and currently supports the Binance exchange. It also includes a simulation mode for testing strategies and algorithms without risking real funds using the Binance testnet.

6.1.2 Enumeration Type Documentation

6.1.2.1 OrderType

```
enum ats::OrderType
```

Enum for different types of orders.

Enumerator

LIMIT	Limit order
MARKET	Market order
STOP_LOSS	Stop loss order
STOP_LOSS_LIMIT	Stop loss limit order
TAKE_PROFIT	Take profit order
TAKE_PROFIT_LIMIT	Take profit limit order
LIMIT_MAKER	Limit maker order
OTCOUNT	Number of order types

6.1.2.2 Side

```
enum ats::Side
```

Enum for buy/sell side of an order.

Enumerator

BUY	Buy side of an order
SELL	Sell side of an order
SCOUNT	Number of sides

6.1.3 Function Documentation

6.1.3.1 OrderTypeToString()

```
std::string ats::OrderTypeToString (
    OrderType t )
```

Converts OrderType enum value to string.

Parameters

<i>t</i>	The OrderType enum value to convert.
----------	--------------------------------------

Returns

A string representation of the OrderType value.

6.1.3.2 SideToString()

```
std::string ats::SideToString (
    Side t )
```

Converts Side enum value to string.

Parameters

<i>t</i>	The Side enum value to convert.
----------	---------------------------------

Returns

A string representation of the Side value.

6.1.3.3 stringToOrderType()

```
OrderType ats::stringToOrderType (  
    const std::string & s )
```

Converts a string to an OrderType enum value.

Parameters

<i>s</i>	The string to convert.
----------	------------------------

Returns

The corresponding OrderType enum value.

6.1.3.4 stringToSide()

```
Side ats::stringToSide (  
    const std::string & s )
```

Converts a string to a Side enum value.

Parameters

<i>s</i>	The string to convert.
----------	------------------------

Returns

The corresponding Side enum value.

Chapter 7

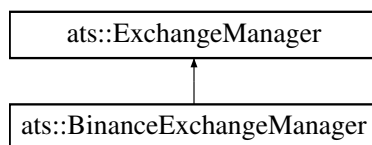
Class Documentation

7.1 ats::BinanceExchangeManager Class Reference

The [BinanceExchangeManager](#) class is an abstract class that defines the interface for managing orders on the Binance exchange.

```
#include <BinanceExchangeManager.h>
```

Inheritance diagram for ats::BinanceExchangeManager:



Public Member Functions

- [BinanceExchangeManager](#) ([OrderManager](#) &orderManager, bool isSimulation=true, std::string api_key="", std::string secret_key="")
Constructor for [BinanceExchangeManager](#) class.
- ~[BinanceExchangeManager](#) ()
Destructor for [BinanceExchangeManager](#) class.
- void **start** ()
Start the [BinanceExchangeManager](#) thread.
- void **run** ()
The [BinanceExchangeManager](#) processing function.
- void **stop** ()
Stop the [BinanceExchangeManager](#) thread.
- bool **isRunning** ()
Check if the [BinanceExchangeManager](#) thread is running.
- void **sendOrder** ([Order](#) &order) override
Send an order to the Binance exchange.
- void **modifyOrder** ([Order](#) &oldOrder, [Order](#) &newOrder) override
Modify an existing order on the Binance exchange.
- void **cancelOrder** ([Order](#) &order) override

- Cancel an existing order on the Binance exchange.*

 - void `getOrderStatus` (`Order` &order, `Json::Value` &result) override

Gets the status of the given order.
- `std::vector< Order >` `getOpenOrders` () override

Gets all open orders for the current user.
- `std::vector< Trade >` `getTradeHistory` (`std::string` symbol) override

Gets the trade history for the specified symbol.
- double `getPrice` (`std::string` symbol) override

Gets the current price for the specified symbol.
- `Order` `jsonToOrder` (`Json::Value` &result)

Converts a JSON object to an `Order` object.
- `Trade` `jsonToTrade` (`Json::Value` &result)

Converts a JSON object to a `Trade` object.
- void `getUserInfo` (`Json::Value` &result)

Gets the current user's information.

Public Member Functions inherited from `ats::ExchangeManager`

- `ExchangeManager` (`OrderManager` &oms)

Constructor for `ExchangeManager` class.

- virtual `~ExchangeManager` ()=default

Virtual destructor for `ExchangeManager` class.

- virtual void `sendOrder` (`Order` &order)=0

Sends an order to the exchange.

- virtual void `modifyOrder` (`Order` &oldOrder, `Order` &newOrder)=0

Modifies an existing order on the exchange.

- virtual void `cancelOrder` (`Order` &order)=0

Cancel an order on the exchange.

- virtual void `getOrderStatus` (`Order` &order, `Json::Value` &result)=0

Retrieves the status of an order on the exchange.

- virtual `std::vector< Order >` `getOpenOrders` ()=0

Retrieves a list of open orders on the exchange.

- virtual `std::vector< Trade >` `getTradeHistory` (`std::string` symbol)=0

Retrieves the trade history for a given symbol on the exchange.

- virtual double `getPrice` (`std::string` symbol)=0

Retrieves the current price for a given symbol on the exchange.

Additional Inherited Members

Protected Attributes inherited from `ats::ExchangeManager`

- `OrderManager` & `mOrderManager`

7.1.1 Detailed Description

The `BinanceExchangeManager` class is an abstract class that defines the interface for managing orders on the Binance exchange.

7.1.2 Constructor & Destructor Documentation

7.1.2.1 BinanceExchangeManager()

```
ats::BinanceExchangeManager::BinanceExchangeManager (
    OrderManager & orderManager,
    bool isSimulation = true,
    std::string api_key = "",
    std::string secret_key = "" ) [explicit]
```

Constructor for [BinanceExchangeManager](#) class.

Parameters

<i>orderManager</i>	Reference to the OrderManager object.
<i>isSimulation</i>	A boolean indicating whether the exchange is a simulation or not.
<i>api_key</i>	The API key for the Binance exchange account.
<i>secret_key</i>	The secret key for the Binance exchange account.

7.1.3 Member Function Documentation

7.1.3.1 cancelOrder()

```
void ats::BinanceExchangeManager::cancelOrder (
    Order & order ) [override], [virtual]
```

Cancel an existing order on the Binance exchange.

Parameters

<i>order</i>	The Order to be cancelled.
--------------	--

Implements [ats::ExchangeManager](#).

7.1.3.2 getOpenOrders()

```
std::vector< Order > ats::BinanceExchangeManager::getOpenOrders ( ) [override], [virtual]
```

Gets all open orders for the current user.

Returns

A vector of all open orders.

Implements [ats::ExchangeManager](#).

7.1.3.3 getOrderStatus()

```
void ats::BinanceExchangeManager::getOrderStatus (
    Order & order,
    Json::Value & result ) [override], [virtual]
```

Gets the status of the given order.

Parameters

<i>order</i>	The order to check the status of.
<i>result</i>	The JSON object containing the result of the operation.

Implements [ats::ExchangeManager](#).

7.1.3.4 getPrice()

```
double ats::BinanceExchangeManager::getPrice (
    std::string symbol ) [override], [virtual]
```

Gets the current price for the specified symbol.

Parameters

<i>symbol</i>	The symbol to get the price for.
---------------	----------------------------------

Returns

The current price for the specified symbol.

Implements [ats::ExchangeManager](#).

7.1.3.5 getTradeHistory()

```
std::vector< Trade > ats::BinanceExchangeManager::getTradeHistory (
    std::string symbol ) [override], [virtual]
```

Gets the trade history for the specified symbol.

Parameters

<i>symbol</i>	The symbol to get the trade history for.
---------------	--

Returns

A vector of all trades for the specified symbol.

Implements [ats::ExchangeManager](#).

7.1.3.6 getUserInfo()

```
void ats::BinanceExchangeManager::getUserInfo (
    Json::Value & result )
```

Gets the current user's information.

Parameters

<i>result</i>	The JSON object containing the result of the operation.
---------------	---

7.1.3.7 isRunning()

```
bool ats::BinanceExchangeManager::isRunning ( )
```

Check if the [BinanceExchangeManager](#) thread is running.

Returns

A boolean indicating whether the [BinanceExchangeManager](#) thread is running or not.

7.1.3.8 jsonToOrder()

```
Order ats::BinanceExchangeManager::jsonToOrder (
    Json::Value & result )
```

Converts a JSON object to an [Order](#) object.

Parameters

<i>result</i>	The JSON object to convert.
---------------	-----------------------------

Returns

An [Order](#) object created from the JSON object.

7.1.3.9 jsonToTrade()

```
Trade ats::BinanceExchangeManager::jsonToTrade (
    Json::Value & result )
```

Converts a JSON object to a [Trade](#) object.

Parameters

<i>result</i>	The JSON object to convert.
---------------	-----------------------------

Returns

A [Trade](#) object created from the JSON object.

7.1.3.10 modifyOrder()

```
void ats::BinanceExchangeManager::modifyOrder (
    Order & oldOrder,
    Order & newOrder ) [override], [virtual]
```

Modify an existing order on the Binance exchange.

Parameters

<i>oldOrder</i>	The old Order object to be modified.
<i>newOrder</i>	The new Order object.

Implements [ats::ExchangeManager](#).

7.1.3.11 sendOrder()

```
void ats::BinanceExchangeManager::sendOrder (
    Order & order ) [override], [virtual]
```

Send an order to the Binance exchange.

Parameters

<code>order</code>	The Order object to be sent.
--------------------	--

Implements [ats::ExchangeManager](#).

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

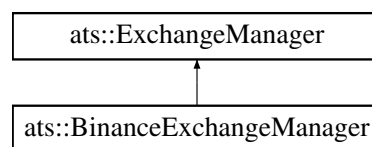
- [BinanceExchangeManager.h](#)
- [BinanceExchangeManager.cpp](#)

7.2 ats::ExchangeManager Class Reference

The [ExchangeManager](#) class is an abstract class that defines the interface for managing orders on an exchange.

```
#include <ExchangeManager.h>
```

Inheritance diagram for ats::ExchangeManager:



Public Member Functions

- [ExchangeManager](#) ([OrderManager](#) &oms)
Constructor for [ExchangeManager](#) class.
- virtual **~ExchangeManager** ()=default
Virtual destructor for [ExchangeManager](#) class.
- virtual void [sendOrder](#) ([Order](#) &order)=0
Sends an order to the exchange.
- virtual void [modifyOrder](#) ([Order](#) &oldOrder, [Order](#) &newOrder)=0
Modifies an existing order on the exchange.
- virtual void [cancelOrder](#) ([Order](#) &order)=0
Cancels an order on the exchange.
- virtual void [getOrderStatus](#) ([Order](#) &order, [Json::Value](#) &result)=0
Retrieves the status of an order on the exchange.
- virtual [std::vector](#)< [Order](#) > [getOpenOrders](#) ()=0
Retrieves a list of open orders on the exchange.
- virtual [std::vector](#)< [Trade](#) > [getTradeHistory](#) ([std::string](#) symbol)=0
Retrieves the trade history for a given symbol on the exchange.
- virtual double [getPrice](#) ([std::string](#) symbol)=0
Retrieves the current price for a given symbol on the exchange.

Protected Attributes

- [OrderManager](#) & mOrderManager

7.2.1 Detailed Description

The [ExchangeManager](#) class is an abstract class that defines the interface for managing orders on an exchange.

7.2.2 Constructor & Destructor Documentation

7.2.2.1 ExchangeManager()

```
ats::ExchangeManager::ExchangeManager (
    OrderManager & oms )
```

Constructor for [ExchangeManager](#) class.

Parameters

<i>oms</i>	A reference to the OrderManager object.
------------	---

7.2.3 Member Function Documentation

7.2.3.1 cancelOrder()

```
virtual void ats::ExchangeManager::cancelOrder (
    Order & order ) [pure virtual]
```

Cancels an order on the exchange.

Parameters

<i>order</i>	The order to be cancelled.
--------------	----------------------------

Implemented in [ats::BinanceExchangeManager](#).

7.2.3.2 getOpenOrders()

```
virtual std::vector< Order > ats::ExchangeManager::getOpenOrders ( ) [pure virtual]
```

Retrieves a list of open orders on the exchange.

Returns

A vector of open orders.

Implemented in [ats::BinanceExchangeManager](#).

7.2.3.3 getOrderStatus()

```
virtual void ats::ExchangeManager::getOrderStatus (
    Order & order,
    Json::Value & result ) [pure virtual]
```

Retrieves the status of an order on the exchange.

Parameters

<i>order</i>	The order to check the status of.
<i>result</i>	A Json::Value object to store the order status information.

Implemented in [ats::BinanceExchangeManager](#).

7.2.3.4 getPrice()

```
virtual double ats::ExchangeManager::getPrice (
    std::string symbol ) [pure virtual]
```

Retrieves the current price for a given symbol on the exchange.

Parameters

<i>symbol</i>	The symbol to retrieve the price for.
---------------	---------------------------------------

Returns

The current price of the symbol.

Implemented in [ats::BinanceExchangeManager](#).

7.2.3.5 getTradeHistory()

```
virtual std::vector< Trade > ats::ExchangeManager::getTradeHistory (
    std::string symbol ) [pure virtual]
```

Retrieves the trade history for a given symbol on the exchange.

Parameters

<i>symbol</i>	The symbol to retrieve the trade history for.
---------------	---

Returns

A vector of [Trade](#) objects representing the trade history.

Implemented in [ats::BinanceExchangeManager](#).

7.2.3.6 modifyOrder()

```
virtual void ats::ExchangeManager::modifyOrder (
    Order & oldOrder,
    Order & newOrder ) [pure virtual]
```

Modifies an existing order on the exchange.

Parameters

<i>oldOrder</i>	The original order.
<i>newOrder</i>	The modified order.

Implemented in [ats::BinanceExchangeManager](#).

7.2.3.7 sendOrder()

```
virtual void ats::ExchangeManager::sendOrder (
    Order & order ) [pure virtual]
```

Sends an order to the exchange.

Parameters

<i>order</i>	The order to be sent to the exchange.
--------------	---------------------------------------

Implemented in [ats::BinanceExchangeManager](#).

7.2.4 Member Data Documentation

7.2.4.1 mOrderManager

`OrderManager& ats::ExchangeManager::mOrderManager [protected]`

A reference to the [OrderManager](#) the EMS will be retrieving orders from

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

- [ExchangeManager.h](#)
- [ExchangeManager.cpp](#)

7.3 ats::MarketData Class Reference

Handles the streaming of market data for the trading system.

```
#include <MarketData.h>
```

Public Member Functions

- [MarketData](#) ([ExchangeManager](#) &ems)
Constructs a new [MarketData](#) object.
- [~MarketData](#) ()
Destroys the [MarketData](#) object.
- [MarketData](#) (const std::vector< std::string > &symbols, [ExchangeManager](#) &ems)
Constructs a new [MarketData](#) object.
- void **start** ()
Starts the market data stream.
- void **run** ()
Runs the market data stream.
- void **stop** ()
Stops the market data stream.
- bool [isRunning](#) ()
Checks whether the market data stream is running.
- void [subscribe](#) (const std::string &symbol)
Subscribes to a symbol for market data.
- void [unsubscribe](#) (const std::string &symbol)
Unsubscribes from a symbol for market data.
- double [getPrice](#) (const std::string &symbol)
Retrieves the current price for a symbol.
- double [getQtyForPrice](#) (const std::string &symbol, double price)
Retrieves the quantity for a given price and symbol.

7.3.1 Detailed Description

Handles the streaming of market data for the trading system.

7.3.2 Constructor & Destructor Documentation

7.3.2.1 `MarketData()` [1/2]

```
ats::MarketData::MarketData (
    ExchangeManager & ems )
```

Constructs a new `MarketData` object.

Parameters

<i>ems</i>	A reference to the ExchangeManager object used to retrieve market data.
------------	---

7.3.2.2 MarketData() [2/2]

```
ats::MarketData::MarketData (
    const std::vector< std::string > & symbols,
    ExchangeManager & ems ) [explicit]
```

Constructs a new [MarketData](#) object.

Parameters

<i>symbols</i>	A vector of symbols to subscribe to for market data.
<i>ems</i>	A reference to the ExchangeManager object used to retrieve market data.

7.3.3 Member Function Documentation**7.3.3.1 getPrice()**

```
double ats::MarketData::getPrice (
    const std::string & symbol )
```

Retrieves the current price for a symbol.

Parameters

<i>symbol</i>	The symbol to retrieve the price for.
---------------	---------------------------------------

Returns

The current price for the symbol.

7.3.3.2 getQtyForPrice()

```
double ats::MarketData::getQtyForPrice (
    const std::string & symbol,
    double price )
```

Retrieves the quantity for a given price and symbol.

Parameters

<i>symbol</i>	The symbol to retrieve the quantity for.
<i>price</i>	The price to retrieve the quantity for.

Returns

The quantity for the given price and symbol.

7.3.3.3 isRunning()

```
bool ats::MarketData::isRunning ( )
```

Checks whether the market data stream is running.

Returns

True if the market data stream is running, false otherwise.

7.3.3.4 subscribe()

```
void ats::MarketData::subscribe (
    const std::string & symbol )
```

Subscribes to a symbol for market data.

Parameters

<i>symbol</i>	The symbol to subscribe to.
---------------	-----------------------------

7.3.3.5 unsubscribe()

```
void ats::MarketData::unsubscribe (
    const std::string & symbol )
```

Unsubscribes from a symbol for market data.

Parameters

<i>symbol</i>	The symbol to unsubscribe from.
---------------	---------------------------------

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

- [MarketData.h](#)
- [MarketData.cpp](#)

7.4 ats::Order Struct Reference

The [Order](#) struct represents an order to be placed on an exchange.

```
#include <OrderManager.h>
```

Public Member Functions

- [Order](#) (long [id](#), [OrderType](#) [type](#), [Side](#) [side](#), std::string [symbol](#), double [quantity](#), double [price](#), double [stopPrice](#)=0., double [icebergQty](#)=0., long [recvWindow](#)=0, long [emsId](#)=0, std::string [timeInForce](#)="")

The [Order](#) constructor.

Public Attributes

- long [id](#)
- std::string [symbol](#)
- double [quantity](#)
- double [price](#)
- [OrderType](#) [type](#)
- [Side](#) [side](#)
- double [stopPrice](#)
- double [icebergQty](#)
- long [recvWindow](#)
- long [emsId](#)
- std::string [timeInForce](#)

7.4.1 Detailed Description

The [Order](#) struct represents an order to be placed on an exchange.

7.4.2 Constructor & Destructor Documentation

7.4.2.1 Order()

```
ats::Order::Order (
    long id,
    OrderType type,
    Side side,
    std::string symbol,
    double quantity,
    double price,
    double stopPrice = 0.,
    double icebergQty = 0.,
    long recvWindow = 0,
    long emsId = 0,
    std::string timeInForce = "" ) [inline]
```

The [Order](#) constructor.

Parameters

<i>id</i>	The order ID.
<i>symbol</i>	The trading symbol of the order.
<i>quantity</i>	The quantity of the asset to be traded in the order.
<i>price</i>	The price per unit of the asset in the order.
<i>type</i>	The type of the order (LIMIT, MARKET, STOP_LOSS, etc.).
<i>side</i>	The side of the order (BUY or SELL).
<i>stopPrice</i>	The stop price of the order (if applicable).
<i>icebergQty</i>	The iceberg quantity of the order (if applicable).
<i>recvWindow</i>	The receive window of the order (if applicable).
<i>emsId</i>	The ID assigned by the EMS to the order (if applicable).
<i>timeInForce</i>	The time in force of the order (if applicable).

7.4.3 Member Data Documentation

7.4.3.1 emsId

```
long ats::Order::emsId
```

The EMS ID of the order.

7.4.3.2 icebergQty

```
double ats::Order::icebergQty
```

The iceberg quantity of the order (only for LIMIT_MAKER orders).

7.4.3.3 id

```
long ats::Order::id
```

The ID of the order.

7.4.3.4 price

```
double ats::Order::price
```

The price of the asset in the quote currency.

7.4.3.5 quantity

```
double ats::Order::quantity
```

The quantity of the asset to buy/sell.

7.4.3.6 recvWindow

```
long ats::Order::recvWindow
```

The receive window of the order (in milliseconds).

7.4.3.7 side

```
Side ats::Order::side
```

The side of the order (e.g. BUY or SELL).

7.4.3.8 stopPrice

```
double ats::Order::stopPrice
```

The stop price of the order (only for STOP_LOSS, STOP_LOSS_LIMIT, TAKE_PROFIT, and TAKE_PROFIT_LIMIT orders).

7.4.3.9 symbol

```
std::string ats::Order::symbol
```

The trading symbol of the order.

7.4.3.10 timeInForce

```
std::string ats::Order::timeInForce
```

The time in force of the order (e.g. GTC, IOC, FOK, etc.).

7.4.3.11 type

```
OrderType ats::Order::type
```

The type of the order (e.g. LIMIT, MARKET, etc.).

The documentation for this struct was generated from the following file:

- [OrderManager.h](#)

7.5 ats::OrderManager Class Reference

A class for managing orders.

```
#include <OrderManager.h>
```

Public Member Functions

- [OrderManager](#) ()
Construct a new [OrderManager](#) object.
- [~OrderManager](#) ()
Destroy the [OrderManager](#) object.
- void [start](#) ()
Start the order manager thread.
- void [run](#) ()
Run the order manager loop.
- void [stop](#) ()
Stop the order manager thread.
- bool [isRunning](#) ()
Check if the order manager is running.
- void [createOrder](#) ([OrderType](#) type, [Side](#) side, std::string symbol, double quantity, double price)
Create a new order and add it to the queue.
- void [processOrder](#) ([Order](#) order)
Process a single order.
- void [processOrders](#) ()
Process all orders in the queue.
- bool [hasOrders](#) ()
Check if there are orders to be sent.
- [Order](#) & [getOldestOrder](#) ()
Get the oldest order from the order queue.

7.5.1 Detailed Description

A class for managing orders.

7.5.2 Constructor & Destructor Documentation

7.5.2.1 OrderManager()

```
ats::OrderManager::OrderManager ( )
```

Construct a new [OrderManager](#) object.

7.5.2.2 ~OrderManager()

```
ats::OrderManager::~~OrderManager ( )
```

Destroy the [OrderManager](#) object.

7.5.3 Member Function Documentation

7.5.3.1 createOrder()

```
void ats::OrderManager::createOrder (
    OrderType type,
    Side side,
    std::string symbol,
    double quantity,
    double price )
```

Create a new order and add it to the queue.

Parameters

<i>type</i>	The type of order
<i>side</i>	The side of the order
<i>symbol</i>	The symbol to trade
<i>quantity</i>	The quantity to trade
<i>price</i>	The price to trade

7.5.3.2 getOldestOrder()

```
Order & ats::OrderManager::getOldestOrder ( )
```

Get the oldest order from the order queue.

Returns

[Order&](#) The oldest order in the queue

7.5.3.3 hasOrders()

```
bool ats::OrderManager::hasOrders ( )
```

Check if there are orders to be sent.

Returns

true if there are orders waiting to be sent
false otherwise

7.5.3.4 isRunning()

```
bool ats::OrderManager::isRunning ( )
```

Check if the order manager is running.

Returns

true if the order manager is running, false otherwise

7.5.3.5 processOrder()

```
void ats::OrderManager::processOrder (
    Order order )
```

Process a single order.

Parameters

<i>order</i>	The order to process
--------------	----------------------

7.5.3.6 processOrders()

```
void ats::OrderManager::processOrders ( )
```

Process all orders in the queue.

7.5.3.7 run()

```
void ats::OrderManager::run ( )
```

Run the order manager loop.

7.5.3.8 start()

```
void ats::OrderManager::start ( )
```

Start the order manager thread.

7.5.3.9 stop()

```
void ats::OrderManager::stop ( )
```

Stop the order manager thread.

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

- [OrderManager.h](#)
- OrderManager.cpp

7.6 ats::Position Struct Reference

Represents a position with its quantity and price.

```
#include <PositionManager.h>
```

Public Member Functions

- **Position** ()
Default constructor that initializes the quantity and price to 0.
- [Position](#) (double q, double p)
Constructor that initializes the quantity and price to the given values.
- double [total](#) ()
Calculates the total value of the position.

Public Attributes

- double **quantity**
Quantity of the position.
- double **price**
Price of the position.

7.6.1 Detailed Description

Represents a position with its quantity and price.

7.6.2 Constructor & Destructor Documentation

7.6.2.1 Position()

```
ats::Position::Position (
    double q,
    double p ) [inline]
```

Constructor that initializes the quantity and price to the given values.

Parameters

q	Quantity of the asset
p	Price of the asset

7.6.3 Member Function Documentation

7.6.3.1 total()

```
double ats::Position::total ( ) [inline]
```

Calculates the total value of the position.

Returns

The total value of the position (quantity * price).

The documentation for this struct was generated from the following file:

- [PositionManager.h](#)

7.7 ats::PositionManager Class Reference

Manages the positions and the PnL of a trading system.

```
#include <PositionManager.h>
```

Public Member Functions

- **PositionManager** ()
Default constructor that initializes the market data to a default instance.
- **PositionManager** ([MarketData](#) &marketData)
Constructor that initializes the market data to the given instance.
- **~PositionManager** ()
Destructor that stops the [PositionManager](#) if it is running.
- void **start** ()
Starts the [PositionManager](#) thread.
- void **run** ()
Runs the [PositionManager](#) loop.
- void **stop** ()
Stops the [PositionManager](#) thread.
- bool **isRunning** ()
Returns whether the [PositionManager](#) is running.
- double **getPnL** ()
Returns the current PnL of the trading system.
- double **getPosition** (std::string symbol)
Returns the current position of the given symbol (quantity held).
- void **updatePosition** (std::string symbol, double quantity)
Updates the position of the given symbol.

7.7.1 Detailed Description

Manages the positions and the PnL of a trading system.

7.7.2 Constructor & Destructor Documentation

7.7.2.1 PositionManager()

```
ats::PositionManager::PositionManager (
    MarketData & marketData )
```

Constructor that initializes the market data to the given instance.

Parameters

<i>marketData</i>	The market data to use.
-------------------	-------------------------

7.7.3 Member Function Documentation

7.7.3.1 getPnL()

```
double ats::PositionManager::getPnL ( )
```

Returns the current PnL of the trading system.

Returns

The current PnL.

7.7.3.2 getPosition()

```
double ats::PositionManager::getPosition (
    std::string symbol )
```

Returns the current position of the given symbol (quantity held).

Parameters

<i>symbol</i>	The symbol of the position to retrieve.
---------------	---

Returns

The current position of the given symbol.

7.7.3.3 isRunning()

```
bool ats::PositionManager::isRunning ( )
```

Returns whether the [PositionManager](#) is running.

Returns

True if the [PositionManager](#) is running, false otherwise.

7.7.3.4 updatePosition()

```
void ats::PositionManager::updatePosition (
    std::string symbol,
    double quantity )
```

Updates the position of the given symbol.

Parameters

<i>symbol</i>	The symbol of the position to update.
<i>quantity</i>	The new quantity of the position.

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

- [PositionManager.h](#)
- [PositionManager.cpp](#)

7.8 ats::RiskManager Class Reference

A class that handles risk management.

```
#include <RiskManager.h>
```

7.8.1 Detailed Description

A class that handles risk management.

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

- [RiskManager.h](#)

7.9 ats::Strategy Class Reference

Defines an abstract interface for trading strategies.

```
#include <Strategy.h>
```

Public Member Functions

- [Strategy](#) (std::string symbol, [MarketData](#) &data, [OrderManager](#) &orderManager, std::vector< double > prices={})
Constructs a [Strategy](#) object.
- virtual ~**Strategy** ()
Destructs the [Strategy](#) object.
- virtual void **start** ()
Starts the strategy thread.
- virtual void **run** ()
Runs the strategy.
- virtual void **stop** ()
Stops the strategy.
- bool [isRunning](#) ()
Returns whether the strategy is running.

Protected Member Functions

- virtual void **updatePrice** ()=0
Updates the historical price vector.
- virtual bool [getSignal](#) ()=0
Gets the trading signal.
- virtual void **buy** ()
Sends a buy order to the [OrderManager](#).
- virtual void **sell** ()
Sends a sell order to the [OrderManager](#).

Protected Attributes

- [MarketData](#) & **mData**
[MarketData](#) object to get market information.
- [OrderManager](#) & **mOrderManager**
[OrderManager](#) object to create and manage orders.
- std::string **mSymbol**
The symbol the strategy is trading.
- std::vector< double > **mPrices**
Vector of historical prices.
- std::thread **mStrategyThread**
Thread for running the strategy.
- bool **mRunning**
Flag indicating if the strategy is running.

7.9.1 Detailed Description

Defines an abstract interface for trading strategies.

7.9.2 Constructor & Destructor Documentation

7.9.2.1 Strategy()

```
ats::Strategy::Strategy (
    std::string symbol,
    MarketData & data,
    OrderManager & orderManager,
    std::vector< double > prices = {} )
```

Constructs a [Strategy](#) object.

Parameters

<i>symbol</i>	The symbol the strategy will trade
<i>data</i>	MarketData object to get market information
<i>orderManager</i>	OrderManager object to create and manage orders
<i>prices</i>	Vector of historical prices (default empty)

7.9.3 Member Function Documentation

7.9.3.1 getSignal()

```
virtual bool ats::Strategy::getSignal ( ) [protected], [pure virtual]
```

Gets the trading signal.

Returns

True if the strategy signals to buy, false if the strategy signals to sell

7.9.3.2 isRunning()

```
bool ats::Strategy::isRunning ( )
```

Returns whether the strategy is running.

Returns

True if the strategy is running, false otherwise

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

- [Strategy.h](#)
- [Strategy.cpp](#)

7.10 ats::Trade Class Reference

Class representing a trade executed on an exchange.

```
#include <Trade.h>
```

Public Member Functions

- [Trade](#) (long id_, double price_, double quantity_, double quoteQty_, long time_, bool isBuyerMaker_, bool isBestMatch_)

Constructor for [Trade](#) class.

7.10.1 Detailed Description

Class representing a trade executed on an exchange.

7.10.2 Constructor & Destructor Documentation

7.10.2.1 Trade()

```
ats::Trade::Trade (
    long id_,
    double price_,
    double quantity_,
    double quoteQty_,
    long time_,
    bool isBuyerMaker_,
    bool isBestMatch_ )
```

Constructor for [Trade](#) class.

Parameters

<i>id_</i>	Trade ID
<i>price_</i>	Trade price
<i>quantity_</i>	Trade quantity
<i>quoteQty_</i>	Trade quote quantity
<i>time_</i>	Trade execution time
<i>isBuyer↔ Maker_</i>	Flag indicating whether the buyer is the maker
<i>isBestMatch↔ _</i>	Flag indicating whether the trade was the best price match at the time

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

- [Trade.h](#)
- Trade.cpp

Chapter 8

File Documentation

8.1 ats.h File Reference

Main header file for the ATS library.

```
#include "MarketData.h"
#include "Strategy.h"
#include "PositionManager.h"
#include "OrderManager.h"
#include "ExchangeManager.h"
#include "BinanceExchangeManager.h"
#include "RiskManager.h"
```

Namespaces

- namespace [ats](#)

Namespace for the algorithmic trading system.

8.1.1 Detailed Description

Main header file for the ATS library.

Author

Anouar Achghaf

Date

12/02/2023

8.2 ats.h

[Go to the documentation of this file.](#)

```
00001 //
00002 // Created by Anouar Achghaf on 12/02/2023.
00003 //
00011 #ifndef ALGO_TRADING_ATS_H
00012 #define ALGO_TRADING_ATS_H
00013
00014 #include "MarketData.h"
00015 #include "Strategy.h"
00016 #include "PositionManager.h"
00017 #include "OrderManager.h"
00018 #include "ExchangeManager.h"
00019 #include "BinanceExchangeManager.h"
00020 #include "RiskManager.h"
00021
00040 #endif //ALGO_TRADING_ATS_H
```

8.3 BinanceExchangeManager.h File Reference

This class is responsible for managing orders and interacting with the Binance exchange API. This class implements the ExchangeManager interface and provides functionality to send, modify, and cancel orders, as well as get the status of open orders, get the trade history, and get the current price of a symbol. It also provides a method to get the user's account information from the Binance API.

```
#include "ExchangeManager.h"
#include "thread"
#include "binance.h"
#include "json/json.h"
#include "binance_logger.h"
```

Classes

- class [ats::BinanceExchangeManager](#)

The [BinanceExchangeManager](#) class is an abstract class that defines the interface for managing orders on the Binance exchange.

Namespaces

- namespace [ats](#)

Namespace for the algorithmic trading system.

8.3.1 Detailed Description

This class is responsible for managing orders and interacting with the Binance exchange API. This class implements the ExchangeManager interface and provides functionality to send, modify, and cancel orders, as well as get the status of open orders, get the trade history, and get the current price of a symbol. It also provides a method to get the user's account information from the Binance API.

Author

Anouar Achghaf

Date

24/02/2023

Note

This class requires an active Binance API key and secret key to function properly, it is assumed that the spot keys reside in \$HOME/.binance/key and \$HOME/.binance/secret and that the spot testnet keys are in \$HOME/.binance/test_key and \$HOME/.binance/test_secret

8.4 BinanceExchangeManager.h

[Go to the documentation of this file.](#)

```

00001
00015 #ifndef ATS_BINANCEEXCHANGEMANAGER_H
00016 #define ATS_BINANCEEXCHANGEMANAGER_H
00017
00018 #include "ExchangeManager.h"
00019 #include "thread"
00020 #include "binance.h"
00021 #include "json/json.h"
00022 #include "binance_logger.h"
00023
00024 namespace ats {
00025     using namespace binance;
00026
00031     class BinanceExchangeManager : public ExchangeManager {
00032     private:
00033         Server mServer;
00034         Market mMarket;
00035         Account mAccount;
00036         bool mIsSimulation;
00037         bool mRunning;
00038         std::thread mExchangeManagerThread;
00039         std::map<long, long> omsToEmsId, emsToOmsId;
00040
00041     public:
00050         explicit BinanceExchangeManager(OrderManager &orderManager, bool isSimulation = true,
std::string api_key = "",
00051                                     std::string secret_key = "");
00052
00056         ~BinanceExchangeManager();
00057
00061         void start();
00062
00066         void run();
00067
00071         void stop();
00072
00078         bool isRunning();
00079
00085         void sendOrder(Order &order) override;
00086
00093         void modifyOrder(Order &oldOrder, Order &newOrder) override;
00094
00100         void cancelOrder(Order &order) override;
00101
00108         void getOrderStatus(Order &order, Json::Value &result) override;
00109
00115         std::vector<Order> getOpenOrders() override;
00116
00123         std::vector<Trade> getTradeHistory(std::string symbol) override;
00124
00131         double getPrice(std::string symbol) override;
00132
00139         Order jsonToOrder(Json::Value &result);
00140
00147         Trade jsonToTrade(Json::Value &result);
00148
00154         void getUserInfo(Json::Value &result);
00155     };
00156
00157 } // ats
00158
00159 #endif //ATS_BINANCEEXCHANGEMANAGER_H

```

8.5 ExchangeManager.h File Reference

Contains an abstract class ExchangeManager The ExchangeManager class is an abstract class that defines the interface for managing orders on an exchange.

```
#include <thread>
#include <mutex>
#include <queue>
#include "OrderManager.h"
#include "Trade.h"
#include "json/json.h"
```

Classes

- class [ats::ExchangeManager](#)

The [ExchangeManager](#) class is an abstract class that defines the interface for managing orders on an exchange.

Namespaces

- namespace [ats](#)

Namespace for the algorithmic trading system.

8.5.1 Detailed Description

Contains an abstract class ExchangeManager The ExchangeManager class is an abstract class that defines the interface for managing orders on an exchange.

Author

Anouar Achghaf

Date

24/02/2023

8.6 ExchangeManager.h

[Go to the documentation of this file.](#)

```
00001
00012 #ifndef ATS_EXCHANGEMANAGER_H
00013 #define ATS_EXCHANGEMANAGER_H
00014
00015 #include <thread>
00016 #include <mutex>
00017 #include <queue>
00018 #include "OrderManager.h"
00019 #include "Trade.h"
00020 #include "json/json.h"
00021
00022 namespace ats {
00027     class ExchangeManager {
00028     protected:
00029         OrderManager &mOrderManager;
```

```

00031     public:
00032
00038         ExchangeManager(OrderManager &oms);
00039
00043         virtual ~ExchangeManager() = default;
00044
00050         virtual void sendOrder(Order &order) = 0;
00051
00058         virtual void modifyOrder(Order &oldOrder, Order &newOrder) = 0;
00059
00065         virtual void cancelOrder(Order &order) = 0;
00066
00073         virtual void getOrderStatus(Order &order, Json::Value &result) = 0;
00074
00080         virtual std::vector<Order> getOpenOrders() = 0;
00081
00088         virtual std::vector<Trade> getTradeHistory(std::string symbol) = 0;
00089
00096         virtual double getPrice(std::string symbol) = 0;
00097     };
00098
00099 }
00100 #endif //ATS_EXCHANGEMANAGER_H

```

8.7 MarketData.h File Reference

Contains the declaration of the MarketData class, which handles the streaming of market data for the trading system.

```

#include <thread>
#include <mutex>
#include <unordered_map>
#include <unordered_set>
#include "ExchangeManager.h"

```

Classes

- class [ats::MarketData](#)
Handles the streaming of market data for the trading system.

Namespaces

- namespace [ats](#)
Namespace for the algorithmic trading system.

8.7.1 Detailed Description

Contains the declaration of the MarketData class, which handles the streaming of market data for the trading system.

Author

Anouar Achghaf

Date

12/02/2023

8.8 MarketData.h

[Go to the documentation of this file.](#)

```

00001
00008 #ifndef ATS_MARKETDATA_H
00009 #define ATS_MARKETDATA_H
00010 #include <thread>
00011 #include <mutex>
00012 #include <unordered_map>
00013 #include <unordered_set>
00014 #include "ExchangeManager.h"
00015
00016 namespace ats {
00020     class MarketData {
00021     private:
00022         std::thread mMarketDataThread;
00023         std::mutex mDataMutex;
00024         bool mRunning;
00025         std::unordered_set<std::string> mSymbols;
00026         std::unordered_map<std::string, std::vector<double>> mPrices;
00027         ExchangeManager& mExchangeManager;
00029     public:
00034         MarketData(ExchangeManager& ems);
00035
00039         ~MarketData();
00040
00046         explicit MarketData(const std::vector<std::string>& symbols, ExchangeManager& ems);
00047
00051         void start();
00052
00056         void run();
00057
00061         void stop();
00062
00067         bool isRunning();
00068
00073         void subscribe(const std::string& symbol);
00074
00079         void unsubscribe(const std::string& symbol);
00080
00086         double getPrice(const std::string& symbol);
00087
00094         double getQtyForPrice(const std::string& symbol, double price);
00095
00096     private:
00101         void updatePrice(const std::string& symbol);
00102
00106         void updatePrices();
00107
00108     };
00109
00110 } // ats
00112
00113 #endif //ATS_MARKETDATA_H

```

8.9 OrderManager.h File Reference

Contains the declaration of the OrderManager class and related enums and structs.

```

#include <string>
#include <queue>
#include <thread>
#include <mutex>
#include <map>
#include <vector>

```

Classes

- struct [ats::Order](#)
The *Order* struct represents an order to be placed on an exchange.
- class [ats::OrderManager](#)
A class for managing orders.

Namespaces

- namespace [ats](#)
Namespace for the algorithmic trading system.

Enumerations

- enum [ats::OrderType](#) {
[ats::LIMIT](#) , [ats::MARKET](#) , [ats::STOP_LOSS](#) , [ats::STOP_LOSS_LIMIT](#) ,
[ats::TAKE_PROFIT](#) , [ats::TAKE_PROFIT_LIMIT](#) , [ats::LIMIT_MAKER](#) , [ats::OTCOUNT](#) }
Enum for different types of orders.
- enum [ats::Side](#) { [ats::BUY](#) , [ats::SELL](#) , [ats::SCOUNT](#) }
Enum for buy/sell side of an order.

Functions

- std::string [ats::OrderTypeToString](#) (OrderType t)
Converts OrderType enum value to string.
- OrderType [ats::stringToOrderType](#) (const std::string &s)
Converts a string to an OrderType enum value.
- std::string [ats::SideToString](#) (Side t)
Converts Side enum value to string.
- Side [ats::stringToSide](#) (const std::string &s)
Converts a string to a Side enum value.

8.9.1 Detailed Description

Contains the declaration of the OrderManager class and related enums and structs.

Author

Anouar Achghaf

Date

16/02/2023

8.10 OrderManager.h

[Go to the documentation of this file.](#)

```

00001
00008 #ifndef ATS_ORDERMANAGER_H
00009 #define ATS_ORDERMANAGER_H
00010
00011 #include <string>
00012 #include <queue>
00013 #include <thread>
00014 #include <mutex>
00015 #include <map>
00016 #include <vector>
00017
00018 namespace ats {
00023     enum OrderType {
00024         LIMIT,
00025         MARKET,
00026         STOP_LOSS,
00027         STOP_LOSS_LIMIT,
00028         TAKE_PROFIT,
00029         TAKE_PROFIT_LIMIT,
00030         LIMIT_MAKER,
00031         OTCOUNT
00032     };
00033
00039     std::string OrderTypeToString(OrderType t);
00040
00046     OrderType stringToOrderType(const std::string &s);
00047
00052     enum Side {
00053         BUY,
00054         SELL,
00055         SCOUNT
00056     };
00057
00063     std::string SideToString(Side t);
00064
00070     Side stringToSide(const std::string &s);
00071
00072
00076     struct Order {
00077
00078         long id;
00079         std::string symbol;
00080         double quantity;
00081         double price;
00082         OrderType type;
00083         Side side;
00084         double stopPrice;
00085         double icebergQty;
00086         long recvWindow;
00087         long emsId;
00088         std::string timeInForce;
00104         Order(long id, OrderType type, Side side, std::string symbol, double quantity, double price,
00105              double stopPrice = 0., double icebergQty = 0., long recvWindow = 0, long emsId = 0,
00106              std::string timeInForce = "") {
00107             this->id = id;
00108             this->side = side;
00109             this->symbol = symbol;
00110             this->quantity = quantity;
00111             this->type = type;
00112             this->price = price;
00113             this->stopPrice = stopPrice;
00114             this->icebergQty = icebergQty;
00115             this->recvWindow = recvWindow;
00116             this->emsId = emsId;
00117             this->timeInForce = timeInForce;
00118         }
00119     };
00120
00124     class OrderManager {
00125     private:
00126         std::queue<Order> mOrders;
00127         std::queue<Order> mPendingOrders;
00128         std::thread mOrderManagerThread;
00129         std::mutex mOrderCountMutex;
00130         std::mutex mOrderFetchMutex;
00131         bool mRunning;
00132         long mOrderCount;
00133         std::vector<Order> mSentOrders;
00134     public:
00139         OrderManager();
00140
00145         ~OrderManager();

```



```

00146
00151     void start();
00152
00157     void run();
00158
00163     void stop();
00164
00170     bool isRunning();
00171
00181     void createOrder(OrderType type, Side side, std::string symbol, double quantity, double
price);
00182
00188     void processOrder(Order order);
00189
00194     void processOrders();
00195
00202     bool hasOrders();
00203
00209     Order &getOldestOrder();
00210
00211     private:
00217         int getNewOrderId();
00218     };
00219
00220 } // ats
00221
00222 #endif //ATS_ORDERMANAGER_H

```

8.11 PositionManager.h File Reference

This header file contains the declaration of the PositionManager class, which is responsible for managing and tracking the open positions and PnL of a trading strategy. The PositionManager class provides functionality for starting and stopping a separate thread for updating the open positions and PnL, as well as updating the position for a specific symbol.

```

#include <thread>
#include <vector>
#include <unordered_map>
#include <mutex>
#include "MarketData.h"

```

Classes

- struct [ats::Position](#)
Represents a position with its quantity and price.
- class [ats::PositionManager](#)
Manages the positions and the PnL of a trading system.

Namespaces

- namespace [ats](#)
Namespace for the algorithmic trading system.

8.11.1 Detailed Description

This header file contains the declaration of the PositionManager class, which is responsible for managing and tracking the open positions and PnL of a trading strategy. The PositionManager class provides functionality for starting and stopping a separate thread for updating the open positions and PnL, as well as updating the position for a specific symbol.

Author

Anouar Achghaf

Date

15/02/2023

8.12 PositionManager.h

[Go to the documentation of this file.](#)

```
00001 //
00002 // Created by Anouar Achghaf on 15/02/2023.
00003 //
00012 #ifndef ATS_POSITIONMANAGER_H
00013 #define ATS_POSITIONMANAGER_H
00014 #include <thread>
00015 #include <vector>
00016 #include <unordered_map>
00017 #include <mutex>
00018 #include "MarketData.h"
00019
00020 namespace ats {
00021
00025     struct Position{
00026         double quantity;
00027         double price;
00028
00032         Position() : quantity(0), price(0) {}
00033
00039         Position(double q, double p) : quantity(q), price(p) {}
00040
00045         double total() {
00046             return quantity * price;
00047         }
00048     };
00049
00053     class PositionManager {
00054     private:
00055         MarketData &mData;
00056         std::thread mPositionManagerThread;
00057         double mPnL;
00058         std::unordered_map<std::string, Position> mOpenPositions;
00059         bool mRunning;
00060         std::mutex mPositionMutex;
00061     public:
00062
00066         PositionManager();
00067
00072         PositionManager(MarketData &marketData);
00073
00077         ~PositionManager();
00078
00082         void start();
00083
00087         void run();
00088
00092         void stop();
00093
00098         bool isRunning();
00099
00104         double getPnL();
00105
00111         double getPosition(std::string symbol);
00112
00118         void updatePosition(std::string symbol, double quantity);
```

```

00119
00120     private:
00121         void updatePnL();
00122     };
00123
00124 } // ats
00125
00126 #endif //ATS_POSITIONMANAGER_H

```

8.13 RiskManager.h File Reference

Classes

- class [ats::RiskManager](#)
A class that handles risk management.

Namespaces

- namespace [ats](#)
Namespace for the algorithmic trading system.

8.13.1 Detailed Description

Author

Anouar Achghaf

Date

03/03/2023 Contains the declaration of the RiskManager class

8.14 RiskManager.h

[Go to the documentation of this file.](#)

```

00001 //
00002 // Created by Anouar Achghaf on 03/03/2023.
00003 //
00011 #ifndef ATS_RISKMANAGER_H
00012 #define ATS_RISKMANAGER_H
00013
00014 namespace ats {
00015     class RiskManager {
00016         // TODO: Implement RMS
00017     };
00018 } // ats
00019
00020 #endif //ATS_RISKMANAGER_H

```

8.15 Strategy.h File Reference

Defines an abstract interface for trading strategies.

```

#include <thread>
#include <vector>
#include "MarketData.h"
#include "OrderManager.h"

```

Classes

- class [ats::Strategy](#)
Defines an abstract interface for trading strategies.

Namespaces

- namespace [ats](#)
Namespace for the algorithmic trading system.

8.15.1 Detailed Description

Defines an abstract interface for trading strategies.

Author

Anouar Achghaf

Date

12/02/2023

8.16 Strategy.h

[Go to the documentation of this file.](#)

```
00001 //
00002 // Created by Anouar Achghaf on 12/02/2023.
00003 //
00011 #ifndef ATS_STRATEGY_H
00012 #define ATS_STRATEGY_H
00013 #include <thread>
00014 #include <vector>
00015 #include "MarketData.h"
00016 #include "OrderManager.h"
00017
00018 namespace ats {
00019
00023     class Strategy {
00024     protected:
00025         MarketData& mData;
00026         OrderManager& mOrderManager;
00027         std::string mSymbol;
00028         std::vector<double> mPrices;
00029         std::thread mStrategyThread;
00030         bool mRunning;
00031     public:
00039         Strategy(std::string symbol, MarketData& data, OrderManager& orderManager, std::vector<double>
prices={});
00040
00044         virtual ~Strategy();
00045
00049         virtual void start();
00050
00054         virtual void run();
00055
00059         virtual void stop();
00060
00065         bool isRunning();
00066
00067     protected:
00071         virtual void updatePrice() = 0;
00072
00077         virtual bool getSignal() = 0;
00078
00082         virtual void buy();
00083
00087         virtual void sell();
00088     };
00089
00090
00091 } // ats
00092
00093 #endif //ATS_STRATEGY_H
```

8.17 Trade.h File Reference

Contains the Trade class definition.

Classes

- class [ats::Trade](#)
Class representing a trade executed on an exchange.

Namespaces

- namespace [ats](#)
Namespace for the algorithmic trading system.

8.17.1 Detailed Description

Contains the Trade class definition.

Author

Anouar Achghaf

Date

02/03/2023

8.18 Trade.h

[Go to the documentation of this file.](#)

```
00001
00008 #ifndef ATS_TRADE_H
00009 #define ATS_TRADE_H
00010
00011 namespace ats {
00012
00016     class Trade {
00017     private:
00018         long id;
00019         double price;
00020         double quantity;
00021         double quoteQty;
00022         long time;
00023         bool isBuyerMaker;
00024         bool isBestMatch;
00025     public:
00036         Trade(long id_, double price_, double quantity_, double quoteQty_, long time_,
00037             bool isBuyerMaker_, bool isBestMatch_);
00038     };
00039
00040
00041 } // ats
00042
00043 #endif //ATS_TRADE_H
```


Index

- ~OrderManager
 - ats::OrderManager, 32
- ats, 11
 - BUY, 13
 - LIMIT, 12
 - LIMIT_MAKER, 12
 - MARKET, 12
 - OrderType, 12
 - OrderTypeToString, 13
 - OTCOUNT, 12
 - SCOUNT, 13
 - SELL, 13
 - Side, 12
 - SideToString, 13
 - STOP_LOSS, 12
 - STOP_LOSS_LIMIT, 12
 - stringToOrderType, 14
 - stringToSide, 14
 - TAKE_PROFIT, 12
 - TAKE_PROFIT_LIMIT, 12
- ats.h, 43, 44
- ats::BinanceExchangeManager, 15
 - BinanceExchangeManager, 17
 - cancelOrder, 17
 - getOpenOrders, 17
 - getOrderStatus, 18
 - getPrice, 18
 - getTradeHistory, 18
 - getUserInfo, 19
 - isRunning, 19
 - jsonToOrder, 19
 - jsonToTrade, 20
 - modifyOrder, 20
 - sendOrder, 20
- ats::ExchangeManager, 21
 - cancelOrder, 22
 - ExchangeManager, 22
 - getOpenOrders, 22
 - getOrderStatus, 23
 - getPrice, 23
 - getTradeHistory, 23
 - modifyOrder, 24
 - mOrderManager, 24
 - sendOrder, 24
- ats::MarketData, 25
 - getPrice, 27
 - getQtyForPrice, 27
 - isRunning, 28
 - MarketData, 26, 27
 - subscribe, 28
 - unsubscribe, 28
- ats::Order, 29
 - emsId, 30
 - icebergQty, 30
 - id, 30
 - Order, 29
 - price, 30
 - quantity, 30
 - recvWindow, 31
 - side, 31
 - stopPrice, 31
 - symbol, 31
 - timeInForce, 31
 - type, 31
- ats::OrderManager, 32
 - ~OrderManager, 32
 - createOrder, 33
 - getOldestOrder, 33
 - hasOrders, 33
 - isRunning, 33
 - OrderManager, 32
 - processOrder, 34
 - processOrders, 34
 - run, 34
 - start, 34
 - stop, 34
- ats::Position, 35
 - Position, 35
 - total, 36
- ats::PositionManager, 36
 - getPnL, 37
 - getPosition, 37
 - isRunning, 38
 - PositionManager, 37
 - updatePosition, 38
- ats::RiskManager, 38
- ats::Strategy, 39
 - getSignal, 40
 - isRunning, 40
 - Strategy, 40
- ats::Trade, 41
 - Trade, 41
- BinanceExchangeManager
 - ats::BinanceExchangeManager, 17
- BinanceExchangeManager.h, 44, 45
- BUY
 - ats, 13

- cancelOrder
 - ats::BinanceExchangeManager, 17
 - ats::ExchangeManager, 22
- createOrder
 - ats::OrderManager, 33
- emslid
 - ats::Order, 30
- ExchangeManager
 - ats::ExchangeManager, 22
- ExchangeManager.h, 46
- getOldestOrder
 - ats::OrderManager, 33
- getOpenOrders
 - ats::BinanceExchangeManager, 17
 - ats::ExchangeManager, 22
- getOrderStatus
 - ats::BinanceExchangeManager, 18
 - ats::ExchangeManager, 23
- getPnL
 - ats::PositionManager, 37
- getPosition
 - ats::PositionManager, 37
- getPrice
 - ats::BinanceExchangeManager, 18
 - ats::ExchangeManager, 23
 - ats::MarketData, 27
- getQtyForPrice
 - ats::MarketData, 27
- getSignal
 - ats::Strategy, 40
- getTradeHistory
 - ats::BinanceExchangeManager, 18
 - ats::ExchangeManager, 23
- getUserInfo
 - ats::BinanceExchangeManager, 19
- hasOrders
 - ats::OrderManager, 33
- icebergQty
 - ats::Order, 30
- id
 - ats::Order, 30
- isRunning
 - ats::BinanceExchangeManager, 19
 - ats::MarketData, 28
 - ats::OrderManager, 33
 - ats::PositionManager, 38
 - ats::Strategy, 40
- jsonToOrder
 - ats::BinanceExchangeManager, 19
- jsonToTrade
 - ats::BinanceExchangeManager, 20
- LIMIT
 - ats, 12
- LIMIT_MAKER
 - ats, 12
- MARKET
 - ats, 12
- MarketData
 - ats::MarketData, 26, 27
- MarketData.h, 47, 48
- modifyOrder
 - ats::BinanceExchangeManager, 20
 - ats::ExchangeManager, 24
- mOrderManager
 - ats::ExchangeManager, 24
- Order
 - ats::Order, 29
- OrderManager
 - ats::OrderManager, 32
- OrderManager.h, 48, 50
- OrderType
 - ats, 12
- OrderTypeToString
 - ats, 13
- OTCOUNT
 - ats, 12
- Position
 - ats::Position, 35
- PositionManager
 - ats::PositionManager, 37
- PositionManager.h, 51, 52
- price
 - ats::Order, 30
- processOrder
 - ats::OrderManager, 34
- processOrders
 - ats::OrderManager, 34
- quantity
 - ats::Order, 30
- recvWindow
 - ats::Order, 31
- RiskManager.h, 53
- run
 - ats::OrderManager, 34
- SCOUNT
 - ats, 13
- SELL
 - ats, 13
- sendOrder
 - ats::BinanceExchangeManager, 20
 - ats::ExchangeManager, 24
- Side
 - ats, 12
- side
 - ats::Order, 31
- SideToString
 - ats, 13
- start

- ats::OrderManager, [34](#)
- stop
 - ats::OrderManager, [34](#)
- STOP_LOSS
 - ats, [12](#)
- STOP_LOSS_LIMIT
 - ats, [12](#)
- stopPrice
 - ats::Order, [31](#)
- Strategy
 - ats::Strategy, [40](#)
- Strategy.h, [53](#), [54](#)
- stringToOrderType
 - ats, [14](#)
- stringToSide
 - ats, [14](#)
- subscribe
 - ats::MarketData, [28](#)
- symbol
 - ats::Order, [31](#)
- TAKE_PROFIT
 - ats, [12](#)
- TAKE_PROFIT_LIMIT
 - ats, [12](#)
- timeInForce
 - ats::Order, [31](#)
- total
 - ats::Position, [36](#)
- Trade
 - ats::Trade, [41](#)
- Trade.h, [55](#)
- type
 - ats::Order, [31](#)
- unsubscribe
 - ats::MarketData, [28](#)
- updatePosition
 - ats::PositionManager, [38](#)