ATS

Generated by Doxygen 1.9.6

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:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

File Index

5.1 File List

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

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

t	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

t	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

s	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

s	The string to convert.

Returns

The corresponding Side enum value.

Chapter 7

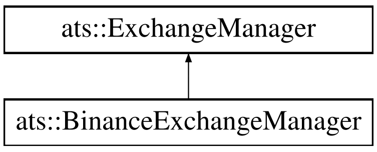
Class Documentation

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

  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.

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

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

order	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

order	The order to check the status of.
result	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

symbol	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

symbol	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

result	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

result	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

result	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

oldOrder	The old Order object to be modified.
newOrder	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

order	The Order object to be sent.

Implements ats::ExchangeManager.

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

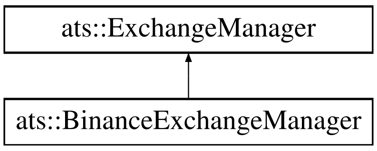
- BinanceExchangeManager.h
- BinanceExchangeManager.cpp

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

oms	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

order	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

order	The order to check the status of.
result	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

symbol	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

symbol	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

oldOrder	The original order.
newOrder	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

order	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

ems	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

symbols	A vector of symbols to subscribe to for market data.
ems	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

symbol	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

symbol	The symbol to retrieve the quantity for.
price	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

symbol	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

symbol	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

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

type	The type of order
side	The side of the order
symbol	The symbol to trade
quantity	The quantity to trade
price	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

order	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

marketData	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

symbol	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

symbol	The symbol of the position to update.
quantity	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

symbol	The symbol the strategy will trade
data	MarketData object to get market information
orderManager	OrderManager object to create and manage orders
prices	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

*id_*	Trade ID
*price_*	Trade price
*quantity_*	Trade quantity
*quoteQty_*	Trade quote quantity
*time_*	Trade execution time
*isBuyer↩ Maker_*	Flag indicating whether the buyer is the maker
isBestMatch↩ _	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.

```
//
// Created by Anouar Achghaf on 12/02/2023.
//
#ifndef ALGO_TRADING_ATS_H
#define ALGO_TRADING_ATS_H

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

#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.

```

#ifndef ATS_BINANCEEXCHANGEMANAGER_H
#define ATS_BINANCEEXCHANGEMANAGER_H

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

namespace ats {
    using namespace binance;

    class BinanceExchangeManager : public ExchangeManager {
    private:
        Server mServer;
        Market mMarket;
        Account mAccount;
        bool mIsSimulation;
        bool mRunning;
        std::thread mExchangeManagerThread;
        std::map<long, long> omsToEmsId, emsToOmsId;

    public:
        explicit BinanceExchangeManager(OrderManager &orderManager, bool isSimulation = true, std::string api_key = "",
                                        std::string secret_key = "");

        ~BinanceExchangeManager();

        void start();

        void run();

        void stop();

        bool isRunning();

        void sendOrder(Order &order) override;

        void modifyOrder(Order &oldOrder, Order &newOrder) override;

        void cancelOrder(Order &order) override;

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

        std::vector<Order> getOpenOrders() override;

        std::vector<Trade> getTradeHistory(std::string symbol) override;

        double getPrice(std::string symbol) override;

        Order jsonToOrder(Json::Value &result);

        Trade jsonToTrade(Json::Value &result);

        void getUserInfo(Json::Value &result);
    };

} // ats

#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.

```

#ifndef ATS_EXCHANGEMANAGER_H
#define ATS_EXCHANGEMANAGER_H

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

namespace ats {
    class ExchangeManager {
    protected:
        OrderManager &mOrderManager;
    public:

        ExchangeManager(OrderManager &oms);

        virtual ~ExchangeManager() = default;

        virtual void sendOrder(Order &order) = 0;

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

        virtual void cancelOrder(Order &order) = 0;

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

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

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

        virtual double getPrice(std::string symbol) = 0;
    };

}
#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.

```
#ifndef ATS_MARKETDATA_H
#define ATS_MARKETDATA_H
#include <thread>
#include <mutex>
#include <unordered_map>
#include <unordered_set>
#include "ExchangeManager.h"

namespace ats {
    class MarketData {
    private:
        std::thread mMarketDataThread;
        std::mutex mDataMutex;
        bool mRunning;
        std::unordered_set<std::string> mSymbols;
        std::unordered_map<std::string, std::vector<double» mPrices;
        ExchangeManager& mExchangeManager;
    public:
        MarketData(ExchangeManager& ems);

        ~MarketData();

        explicit MarketData(const std::vector<std::string>& symbols, ExchangeManager& ems);

        void start();

        void run();

        void stop();

        bool isRunning();

        void subscribe(const std::string& symbol);

        void unsubscribe(const std::string& symbol);

        double getPrice(const std::string& symbol);

        double getQtyForPrice(const std::string& symbol, double price);

    private:
        void updatePrice(const std::string& symbol);

        void updatePrices();

    };


} // ats

#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.

```

#ifndef ATS_ORDERMANAGER_H
#define ATS_ORDERMANAGER_H

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

namespace ats {
    enum OrderType {
        LIMIT,
        MARKET,
        STOP_LOSS,
        STOP_LOSS_LIMIT,
        TAKE_PROFIT,
        TAKE_PROFIT_LIMIT,
        LIMIT_MAKER,
        OTCOUNT
    };

    std::string OrderTypeToString(OrderType t);

    OrderType stringToOrderType(const std::string &s);

    enum Side {
        BUY,
        SELL,
        SCOUNT
    };

    std::string SideToString(Side t);

    Side stringToSide(const std::string &s);


    struct Order {

        long id;
        std::string symbol;
        double quantity;
        double price;
        OrderType type;
        Side side;
        double stopPrice;
        double icebergQty;
        long recvWindow;
        long emsId;
        std::string timeInForce;
        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 = "") {
            this->id = id;
            this->side = side;
            this->symbol = symbol;
            this->quantity = quantity;
            this->type = type;
            this->price = price;
            this->stopPrice = stopPrice;
            this->icebergQty = icebergQty;
            this->recvWindow = recvWindow;
            this->emsId = emsId;
            this->timeInForce = timeInForce;
        }
    };

    class OrderManager {
    private:
        std::queue<Order> mOrders;
        std::queue<Order> mPendingOrders;
        std::thread mOrderManagerThread;
        std::mutex mOrderCountMutex;
        std::mutex mOrderFetchMutex;
        bool mRunning;
        long mOrderCount;
        std::vector<Order> mSentOrders;
    public:
        OrderManager();

        ~OrderManager();

        void start();

        void run();

        void stop();

        bool isRunning();

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

        void processOrder(Order order);

        void processOrders();

        bool hasOrders();

        Order &getOldestOrder();

    private:
        int getNewOrderId();
    };

} // ats

#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.

```
//
// Created by Anouar Achghaf on 15/02/2023.
//
#ifndef ATS_POSITIONMANAGER_H
#define ATS_POSITIONMANAGER_H
#include <thread>
#include <vector>
#include <unordered_map>
#include <mutex>
#include "MarketData.h"

namespace ats {

    struct Position{
        double quantity;
        double price;

        Position() : quantity(0), price(0) {}

        Position(double q, double p) : quantity(q), price(p) {}

        double total() {
            return quantity * price;
        }
    };

    class PositionManager {
    private:
        MarketData &mData;
        std::thread mPositionManagerThread;
        double mPnL;
        std::unordered_map<std::string, Position> mOpenPositions;
        bool mRunning;
        std::mutex mPositionMutex;
    public:

        PositionManager();

        PositionManager(MarketData &marketData);

        ~PositionManager();

        void start();

        void run();

        void stop();

        bool isRunning();

        double getPnL();

        double getPosition(std::string symbol);

        void updatePosition(std::string symbol, double quantity);
    private:
        void updatePnL();
    };


} // ats

#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.

```
//
// Created by Anouar Achghaf on 03/03/2023.
//
#ifndef ATS_RISKMANAGER_H
#define ATS_RISKMANAGER_H

namespace ats {
    class RiskManager {
        // TODO: Implement RMS
    };

} // ats

#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.

```
//
// Created by Anouar Achghaf on 12/02/2023.
//
#ifndef ATS_STRATEGY_H
#define ATS_STRATEGY_H
#include <thread>
#include <vector>
#include "MarketData.h"
#include "OrderManager.h"

namespace ats {

    class Strategy {
    protected:
        MarketData& mData;
        OrderManager& mOrderManager;
        std::string mSymbol;
        std::vector<double> mPrices;
        std::thread mStrategyThread;
        bool mRunning;
    public:
        Strategy(std::string symbol, MarketData& data, OrderManager& orderManager, std::vector<double>
prices={});

        virtual ~Strategy();

        virtual void start();

        virtual void run();

        virtual void stop();

        bool isRunning();

    protected:
        virtual void updatePrice() = 0;

        virtual bool getSignal() = 0;

        virtual void buy();

        virtual void sell();
    };


} // ats

#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.

```
#ifndef ATS_TRADE_H
#define ATS_TRADE_H

namespace ats {

    class Trade {
    private:
        long id;
        double price;
        double quantity;
        double quoteQty;
        long time;
        bool isBuyerMaker;
        bool isBestMatch;
    public:
        Trade(long id_, double price_, double quantity_, double quoteQty_, long time_,
              bool isBuyerMaker_, bool isBestMatch_);
    };


} // ats

#endif //ATS_TRADE_H
```

Index