

tbq-Quant

algorithmic trading platform

March 2013
draft beta document
tbq-Quant v1.1beta3



www.thebonnotgang.com

*I was cautioned to surrender, this I could not do;
I took my gun and vanished.*
The Partisan - Leonard Cohen

Index

Introduction	5
Mission.....	6
KISS	7
Create, Test, Trade.....	7
Prerequisites	7
Functionalities	8
Download	9
Logical Data Model.....	10
System Use Case Scenario.....	11
Support & Help	12
Contacts.....	12
Platform Licence	13
Set-up	14
Download & Install tbq-Quant platform	14
Requirements	14
Downloads	14
Windows Installation.....	15
Linux & Mac OS X Installation	15
Running an example strategy.....	16
Domain Model Components	18
The Model	19
Account	20
Account Implementations	20
Declaration	20
Portfolio	21
Broker	22
Broker Implementations.....	22
Declaration	22
Sending Orders	23
Cancel Order.....	23
Order.....	24
Declaration	25

MarketDataFeed	26
MarketDataFeed Implementations	26
Declaration	27
MarketDataEvent.....	28
TickEvent.....	28
CandleEvent	28
Declaration	28
Strategy	29
Declaration	29
TradingSystem.....	30
Declaration	30
How it all goes together.....	31
Store Service	34
Implementations.....	34
Report Service	35
Implementations.....	35
Declaration	35
Alarm Service.....	35
CEP Provider	36
Implementations.....	36
Declaration	37
Conclusion	38
In Action.....	39
Create a Strategy	40
Pre-Requisites	40
Add the libraries	40
Create your first Strategy	41
A Typical Strategy Structure.....	41
In Action: SimpleMovingAverage crossover	43
What we did.....	47
Test a Strategy	48
In Action: Assessing SimpleMovingAverage crossover strategy.....	48
Trade (execute) a Strategy	53
Appendix	54

Skeleton Strategy Code	55
Simple Moving Averages Crossover Strategy Code	56

Introduction

Advances in technology have greatly changed the financial industry, algorithmic trading is gaining popularity around the world. From the original order processing to the current cutting-edge technology, including trading systems, algorithmic trading has evolved into a billion dollar industry. It change the way financial assets are traded. Every step of the trading process, from order entry to trading venue to back office, is now highly automated, dramatically reducing the costs incurred by intermediaries. By reducing the frictions and costs of trading, technology has the potential to enable a more efficient risk sharing, facilitate hedging, improve liquidity, and make prices more efficient.

Nowadays all large broker-dealers offer a suite of algorithms to its institutional customers, helping them execute orders in a single stock, in pairs of stocks, or in baskets of stocks. Algorithms typically determine the timing, price, quantity and routing of orders, dynamically monitoring market conditions across different securities and trading venues, reducing market impact by optimally and sometimes randomly breaking large orders into smaller pieces, as well as closely tracking benchmarks, such as the volume-weighted average price (VWAP) over the execution interval. In the pursuit of a desired position, algorithms often use a mix of both active and passive strategies, employing both limit orders and marketable orders.

The tbg-Quant platform enables anyone with an interest in financial markets to access algorithmic trading technologies used by Hedge Funds.

After reading this document, you should have a complete understanding of the tbg-Quant platform. You will be able to create, test and execute your own automatic quantitative trading strategies on either historical or live market data.

Mission

Our mission is to empower traders to create a personal Quant Desk by providing a robust and simple to use algorithmic trading platform.

KISS

We strongly believe in the KISS principle (Keep It Simple Stupid) and we adopt it throughout our processes. We empower the trader to focus on strategy development, testing and final execution without losing valuable time learning the underneath software technology. The tbg-Quant platform consists of a robust yet simple software platform and IT infrastructure, empowering users to trade like a Hedge Fund.

Create, Test, Trade

- [Create](#) strategies using the build-in editor via web/HTML5 or by using your favorite IDE like Eclipse/Net Beans, etc.
- [Test](#) a strategy in the paper trading mode using historical data
- [Test](#) a strategy in the paper trading mode using live market data
- [Trade](#) (execute) a strategy in the live trading mode using live market data

Prerequisites

A basic knowledge of high-level programming languages is necessary in order to write a personal strategy. The framework is 100% pure Java and run on Windows, Linux and Mac OS X.

The Bonnot Gang offers free consultancy to support you in the writing and testing of new strategies. Please see chapter "Support & Help".

Functionalities

Core Framework

- 100% Java
- Scalable and extensible
- Event driven architecture
- Supports Siddhi & Esper CEP
- Basic persistence layer embedded inside the framework
- Add on persistence layer supports
- High Level Trade System class
- *High Level Event Study class **

The core framework of the tbg-Quant platform is very extensible. It is possible to easily add third party code libraries (third-party mathematical analysis software).

tbg-QuantDesk

- Web application interface (HTML5)

Supported Brokers

- Paper Broker
- Interactive Brokers

Market Data Feeds

- Yahoo daily, weekly and monthly
- *Google daily, weekly and monthly **
- CSV custom data
- The Bonnot Gang, 1 minute candles
- Interactive Brokers

(*) Not yet available, contact us for details

Download

tbg-Quant platform can be downloaded from the Bonnot Gang website.

There are 2 main components:

1. **tbg-Quant**

This is the platform's core framework, which provides all basic functionalities.

Examples package is included with source code, a helpful starting point to begin coding a strategy from scratch.

2. **tbg-QuantDesk**

This is the front-end, html5, enabling the creation, testing and execution of new strategies, as well as monitoring their performance.

To run the platform, the tbg-QuantDesk package is not mandatory.

Download Link: <http://www.thebonnotgang.com>

Logical Data Model

The diagram below demonstrates the tbg-Quant platform logical data model.

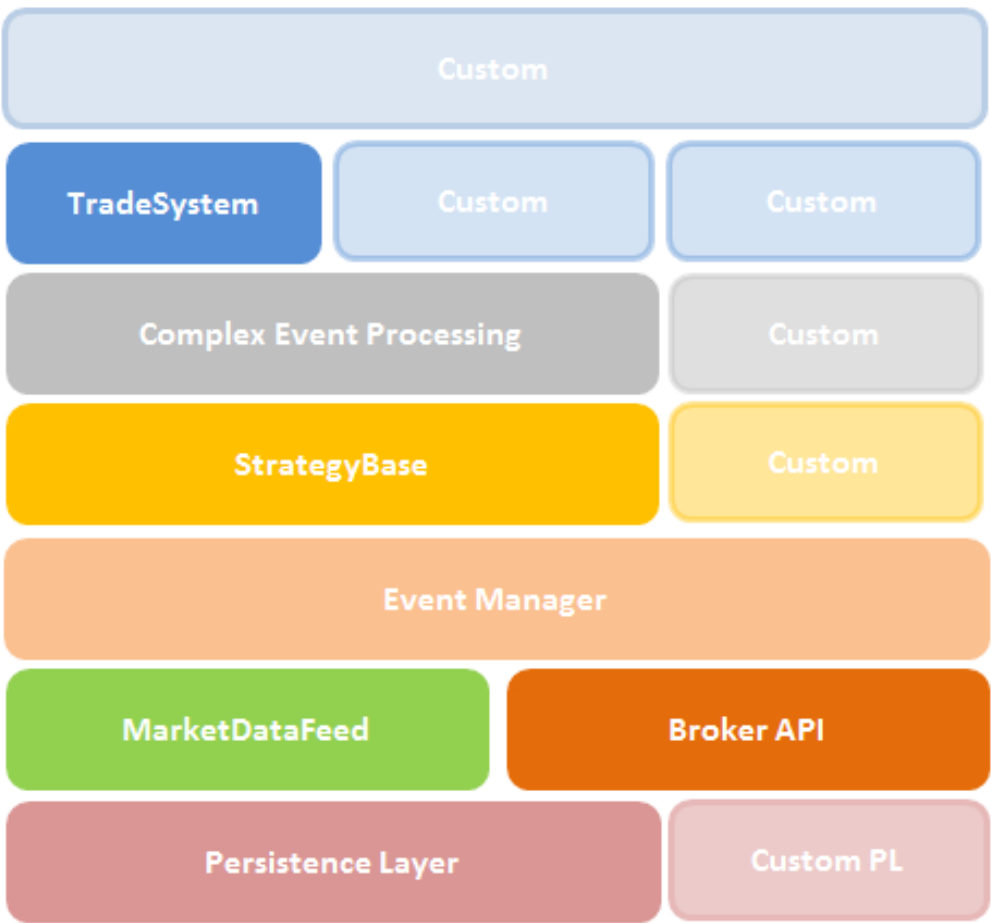


Figure 1 - tbg-Quant logical data model

System Use Case Scenario

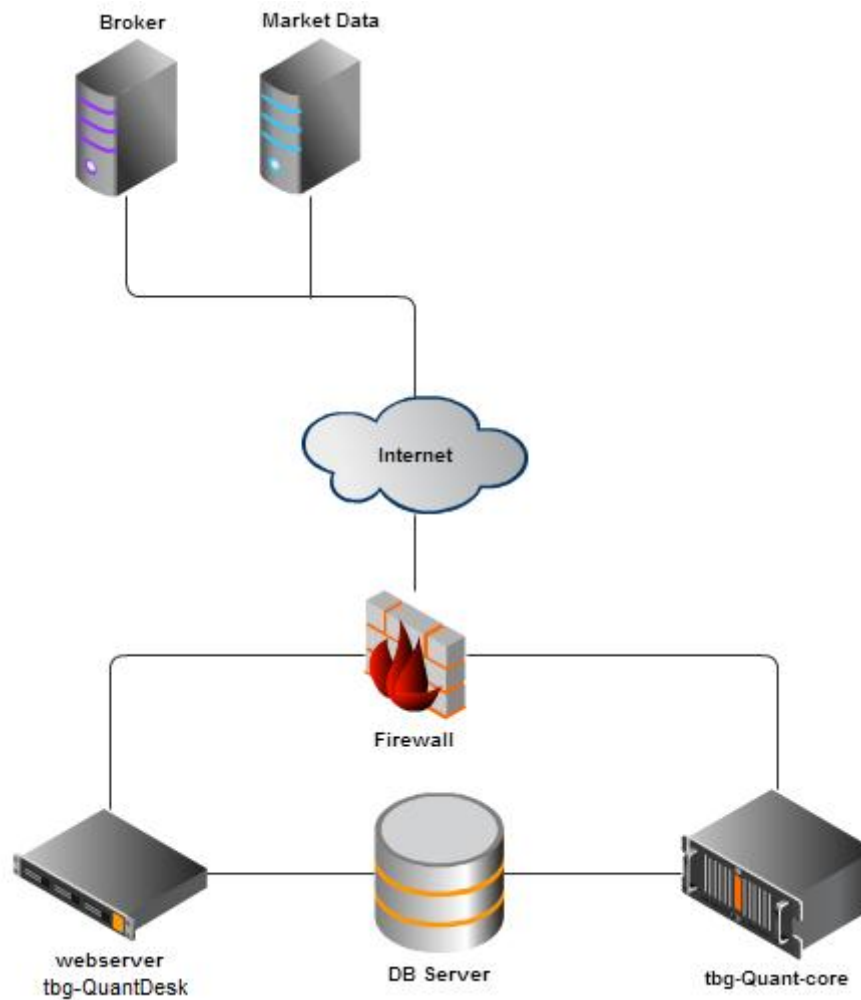


Figure 2 - tbg-Quant Platform scenario

Support & Help

The Bonnot Gang provides **free** assistance in coding and testing strategies. All requests are carefully evaluated and priority is given based on project importance. The Bonnot Gang reserves the right to provide assistance based on its own discretion.

Please contact us to discuss your project needs.

To ensure confidentiality, a non-disclosure agreement will be signed.

A user community is also available to exchange information.

<http://thebonnotgang.com/tbg/support/>

Contacts

The Bonnot Gang

Web : www.thebonnotgang.com

e-maill : info@thebonnotgang.com

Community : <http://thebonnotgang.com/tbg/support/community/>

Platform Licence

TBG-QUANT SOFTWARE LICENSE AGREEMENT

This is a legal agreement between you and TBG-QUANT Software (The Bonnot Gang) covering your use of TBG-QUANT (the "Software").

1. TBG-QUANT is provided as freeware, but only for private, non-commercial use.
 - a. TBG-QUANT is free for educational use (schools and universities) and for use in charity or humanitarian organisations.
 - b. If you intend to use TBG-QUANT at your place of business or for commercial purposes, please contact us by at info@thebonnotgang.com for prices, discounts and payment methods.
2. TBG-QUANT Software is owned by The Bonnot Gang and is protected by copyright laws and international treaty provisions. Therefore, you must treat the Software like any other copyrighted material.
3. You may not distribute, rent, sub-license or otherwise make available to others the Software or documentation or copies thereof, except as expressly permitted in this License without prior written consent from The Bonnot Gang. In the case of an authorized transfer, the transferee must agree to be bound by the terms and conditions of this License Agreement.
4. You may not remove any proprietary notices, labels, trademarks on the Software or documentation. You may not modify, de-compile, disassemble or reverse engineer the Software.
5. Limited warranty: TBG-QUANT and documentation are "as is" without any warranty as to their performance, merchantability or fitness for any particular purpose. The licensee assumes the entire risk as to the quality and performance of the software. In no event shall TBG-QUANT or anyone else who has been involved in the creation, development, production, or delivery of this software be liable for any direct, incidental or consequential damages, such as, but not limited to, loss of anticipated profits, benefits, use, or data resulting from the use of this software, or arising out of any breach of warranty.

Copyright (C) 2013 by The Bonnot Gang (www.thebonnotgang.com), All rights reserved.

Set-up

This chapter covers tbg-Quant platform basic installation and configuration for Windows, Linux and Mac OS X. It does not cover final configuration for live execution (for this purpose please check the online community forums).

Download & Install tbg-Quant platform

Requirements

The tbg-Quant platform requires the SUN Java Virtual Machine (JVM >1.6).

Please ensure that your system enables JVM or go to the following website, download and install the latest JVM:

www.java.com/getjava

tbg-Quant_v1.1 bundles a full Java Runtime Environment.

Downloads

Go to www.thebonnotgang.com and download the following package in accordance with your operating system:

- **tbg-Quant-vXXX-setup.exe** **(Windows)**
- **tbg-Quant-vXXX.tgz** **(Linux / OsX)**

This package includes the following components:

- tbg-Quant core framework
- tbg-Quant examples (including source code)

The tbg-QuantDesk package is not mandatory, but is very useful in monitoring and assessing performances.

Afterwards, download and install the package by following the instruction provided below:

Windows Installation

The tbg-Quant-vXXX-setup.exe package is an auto-installer.
Please follow online instructions.

Linux & Mac OS X Installation

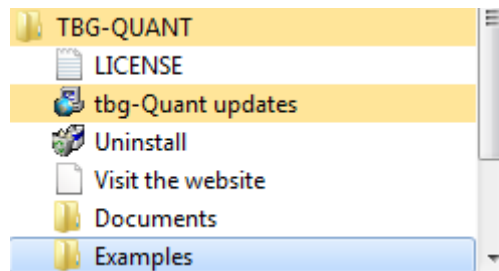
Unzip the tbg-Quant-vXXX.tgz package inside your home directory:

```
$ cd ~/
$ tar -zxvf tbg-Quant-vXXX.tgz
$ cd tbg-Quant
$ ls -l
```

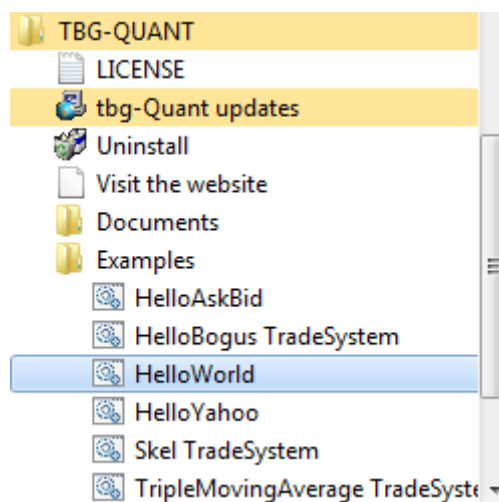

Running an example strategy

Once you installed the platform, make sure that it is set-up correctly.

For the Windows installation, select the start->TBG-QUANT menu, you should see something similar to the following:

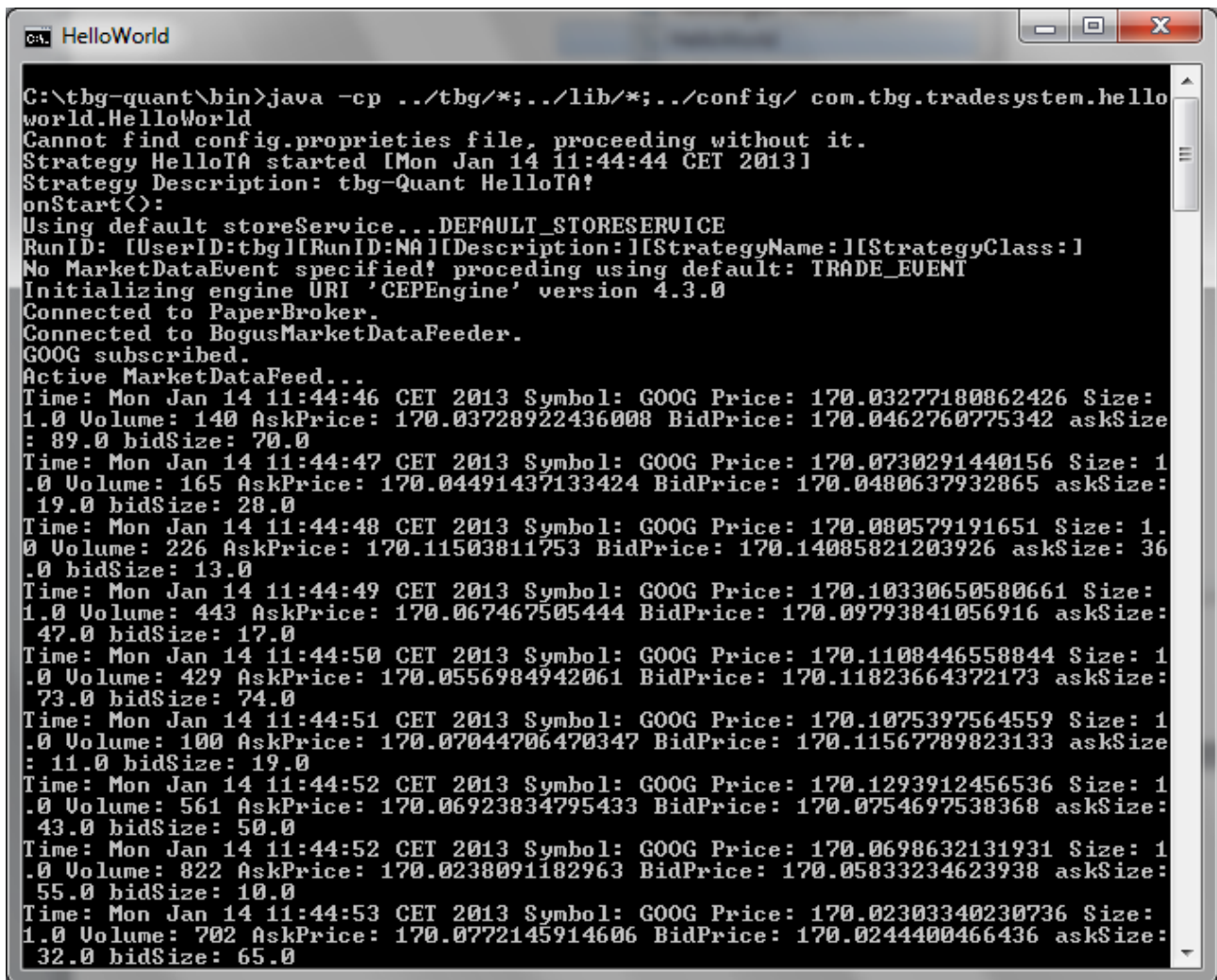


Select the **Examples** folder:



Select the **HelloWorld** example and click.

As soon as you start the example, a command window will appear much like the following image:



```
C:\tbg-quant\bin>java -cp ../tbg/*;../lib/*;../config/ com.tbg.tradesystem.hello
world.HelloWorld
Cannot find config.properties file, proceeding without it.
Strategy HelloTA started [Mon Jan 14 11:44:44 CET 2013]
Strategy Description: tbg-Quant HelloTA!
onStart():
Using default storeService...DEFAULT_STORESERVICE
RunID: [UserID:tbg][RunID:NA][Description:][StrategyName:][StrategyClass:]
No MarketDataEvent specified! proceeding using default: TRADE_EVENT
Initializing engine URI 'CEPEngine' version 4.3.0
Connected to PaperBroker.
Connected to BogusMarketDataFeeder.
GOOG subscribed.
Active MarketDataFeed...
Time: Mon Jan 14 11:44:46 CET 2013 Symbol: GOOG Price: 170.03277180862426 Size:
1.0 Volume: 140 AskPrice: 170.03728922436008 BidPrice: 170.0462760775342 askSize:
89.0 bidSize: 70.0
Time: Mon Jan 14 11:44:47 CET 2013 Symbol: GOOG Price: 170.0730291440156 Size: 1
.0 Volume: 165 AskPrice: 170.04491437133424 BidPrice: 170.0480637932865 askSize:
19.0 bidSize: 28.0
Time: Mon Jan 14 11:44:48 CET 2013 Symbol: GOOG Price: 170.080579191651 Size: 1
.0 Volume: 226 AskPrice: 170.11503811753 BidPrice: 170.14085821203926 askSize: 36
.0 bidSize: 13.0
Time: Mon Jan 14 11:44:49 CET 2013 Symbol: GOOG Price: 170.10330650580661 Size:
1.0 Volume: 443 AskPrice: 170.067467505444 BidPrice: 170.09793841056916 askSize:
47.0 bidSize: 17.0
Time: Mon Jan 14 11:44:50 CET 2013 Symbol: GOOG Price: 170.1108446558844 Size: 1
.0 Volume: 429 AskPrice: 170.0556984942061 BidPrice: 170.11823664372173 askSize:
73.0 bidSize: 74.0
Time: Mon Jan 14 11:44:51 CET 2013 Symbol: GOOG Price: 170.1075397564559 Size: 1
.0 Volume: 100 AskPrice: 170.07044706470347 BidPrice: 170.11567789823133 askSize:
11.0 bidSize: 19.0
Time: Mon Jan 14 11:44:52 CET 2013 Symbol: GOOG Price: 170.1293912456536 Size: 1
.0 Volume: 561 AskPrice: 170.06923834795433 BidPrice: 170.0754697538368 askSize:
43.0 bidSize: 50.0
Time: Mon Jan 14 11:44:52 CET 2013 Symbol: GOOG Price: 170.0698632131931 Size: 1
.0 Volume: 822 AskPrice: 170.0238091182963 BidPrice: 170.05833234623938 askSize:
55.0 bidSize: 10.0
Time: Mon Jan 14 11:44:53 CET 2013 Symbol: GOOG Price: 170.02303340230736 Size:
1.0 Volume: 702 AskPrice: 170.0772145914606 BidPrice: 170.0244400466436 askSize:
32.0 bidSize: 65.0
```

The HelloWorld example starts a TradingSystem that prints raw quotes for the GOOG symbol.

Quotes are fetched from the BogusMarketDataFeeder, which is a bogus simulator producing fake quotes for the Google Inc. (GOOG) symbol.

If you get something like the picture above that means that your tbg-Quant platform installation and configuration is succeeded.

Try to run the **HelloYahoo** example. It will try to connect to the internet and fetch historical data from Yahoo Finance website.

If you are behind a corporate firewall, please configure the proxy values in the Java Control Panel.

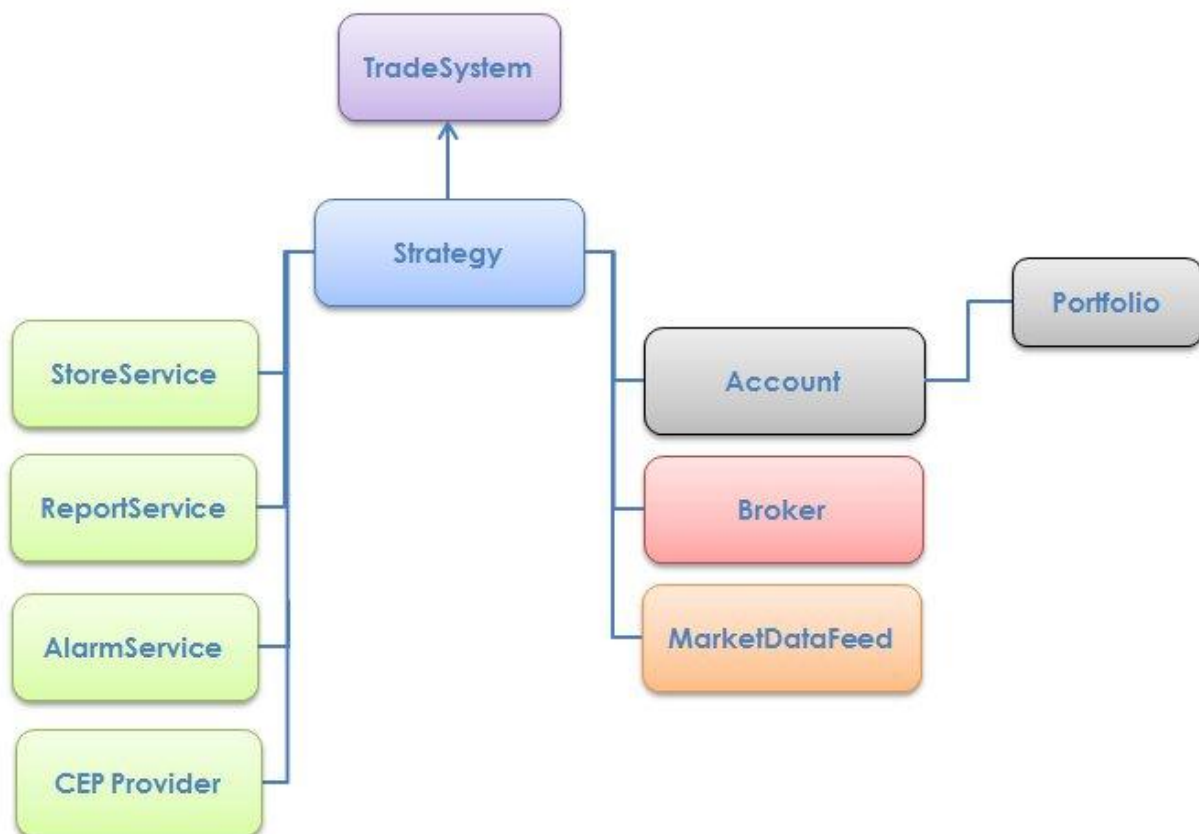
Domain Model Components

After reading this chapter, you will get familiar with the main components of tbg-Quant core framework and how to use them.

There are two categories, shown below:

- **Main Components**
 - Account
 - Portfolio
 - Broker
 - Order
 - MarketDataFeed
 - MarketDataEvent
 - Strategy
 - Trade System
- **Services**
 - StoreService
 - ReportService
 - AlarmService
 - CEP Provider
 - Utils

The Model



|

Account

This represents the trading account.

Main feature:

- **Account ID**
Identify the unique ID of the account.
- **Currency**
The currency of the account
- **Initial Capital**
Is the starting initial capital (cash) available.
- **Cash Balance**
Is the capital (cash) available at a given moment.
- **Invested Capital**
Is the amount of capital (cash) currently invested.
- **Realized PNL (Profit & Loss)**
Is the realized profit & loss made by close investments.
- **Unrealized PNL (Profit & Loss)**
Is the current profit & loss due to a current investment.
- **Equity PNL (Profit & Loss)**
Is the total profit & loss.

Account Implementations

- **PaperAccount**, used for paper trading
- **IBAccount**, InteractiveBrokers account

Declaration

```
// set the PaperAccount, $20000 initial capital  
private final PaperAccount account = new PaperAccount(20000, Currency.USD);
```

```
// set the PaperAccount, €20000 initial capital, default currency is euro  
private final PaperAccount account = new PaperAccount(20000);
```

```
// set the PaperAccount with default initial capital (1ml euro)  
private final PaperAccount account = new PaperAccount();
```

Portfolio

Through Account entity you can access to Portfolio entity which keep tracks of all open positions.

```
account.getPortfolio();
```

Return a list of open positions.

```
account.getPortfolioPosition(symbol);
```

Return the position for the security *symbol*.

Broker

This is the entity that executes orders.

Main features:

- **Connect to the Broker**
- **Disconnect from the Broker**
- **CancelOrder**
- **SendOrder to the Broker**
Sends Order to be fulfilled.

Broker Implementations

- **PaperBroker**
For paper trading.
Possibility to specify commission cost and slippage.
At this stage every order passed to the PaperBroker will be full filled and executed.
PaperBroker handles only MARKET Orders.
- **IBBroker via API**
InteractiveBrokers implementation for live trading.
- **FIX Protocol**
To be implemented, please contact us for more details.

Declaration

```
private final IBroker broker = new PaperBroker(account);
{
    broker.setBrokerCommissions(SecurityType.STK, 10.0);
    broker.setSlippage(0.5);
}

/**
 * Sets the InteractiveBrokersAdapter
 */
private final InteractiveBrokersAdapter interactiveBrokersAdapter = new InteractiveBrokersAdapter();

/**
 * Sets the broker
 */
private final IBroker broker = new IBroker(account, interactiveBrokersAdapter);
{
    broker.setDebug(true);
    broker.setBrokerId(1);
}
```

Sending Orders

```
// Open position
Order order = new Order();
order.setSecurity(getSecurityBySymbol("IBM"));
order.setOrderSide(OrderSide.BUY);
order.setOrderType(OrderType.MARKET);
order.setQuantity(100);

broker.sendOrder(order);
```

Sending a MARKET order to BUY 100 shares of IBM.

Cancel Order

```
broker.cancelOrder(orderId);
```

Sending a CANCEL ORDER to the broker.

You can access this functionality using the TradingSystem high-level method:

```
cancelOrderForSymbol(security.getSymbol());
```


Order

An order is an instruction to a Broker to BUY or SELL a security.

Attributes:

- **Order ID**
Unique ID of an order.
- **Order Type**
`MARKET` ,
`LIMIT`,
`STOP`,
`STOP_LIMIT`
`TRAILING_STOP`
- **Security**
- **Order Side**
`BUY`
`SELL`
`SHORT_SELL`
- **Time In Force**
Specify how long the order will remain active before it is executed or expires.
- **Target TimeStamp**
- **Expiration TimeStamp**
- **Quantity**
Number of shares to BUY/SELL
- **Limit Price**
- **Stop Price**
- **Trailing Distance**
- **Order Status**
`NEW`, Order is new, never submitted to the broker.
`PENDING_SUBMIT`, Order has been sent to the broker.
`PROCESSING`, Order has been submitted by the broker and is now processed.
`FILLED`, Order is filled, only if is complete. Partial filling is not considered a filled order. This is a terminal state.
`REJECTED`, Order has been rejected by the broker. This is a terminal state.
`CANCELED`, Order cancelled. Order may still be partially filled.
`PENDING_CANCEL`, User requested to cancel this Order. Broker received the request, but is still evaluating the request.
`EXPIRED`, Order has expired.

Declaration

```
Order order = new Order();  
order.setSecurity(security);  
order.setOrderSide(OrderSide.BUY);  
order.setOrderType(OrderType.MARKET);  
order.setQuantity(10.0);  
order.setTimeInForce(OrderTIF.DAY);
```

Market Order, valid for the day, BUY 10 shares of security.

```
Order order = new Order();  
order.setSecurity(security);  
order.setOrderSide(OrderSide.SHORT_SELL);  
order.setOrderType(OrderType.MARKET);  
order.setQuantity(1000);  
order.setTimeInForce(OrderTIF.DAY);
```

Market Order, valid for the day, SELL SHORT 1000 shares of security.

MarketDataFeed

This component provides market data, both historical or live data.

Main features:

- Connect to MarketDataFeed
- Disconnect from MarketDataFeed
- **Subscribe market data for a security**
Require a data subscription for a given security
- **Activate a market data subscription**
Activate the subscriptions and begin to receive market data
- **Set the market data Event**
Set which type of market event we should expect.
There are two MarketEvent: the [TickEvent](#) and the [CandleEvent](#).

MarketDataFeed Implementations

- **BogusMarketDataFeed**
Provides random fake data.
Supports TickEvents and CandleEvents.
- **YahooMarketDataFeed**
Provides historical data from Yahoo Finance.
[DAILY](#), [WEEKLY](#), [MONTHLY](#)
CandleEvents only
- **GoogleMarketDataFeed** (* not available in beta release)
Provides historical data from Google Finance.
[DAILY](#), [WEEKLY](#), [MONTHLY](#)
CandleEvents only
- **TBGMarketDataFeed**
Provides original 1 minute candles from theBonnotGang historical database.
[1 MINUTE CANDLES](#)
Supports original CandleEvent and TickEvents (built on 1 minute candles).
- **IBMarketDataFeed**
Provides interface to InteractiveBrokers live market data.
Supports TickEvents.

Declaration

```
private final IMarketDataFeed marketDataFeed = new BogusMarketDataFeed();
```

Bogus Feed.

```
private final YahooMarketDataFeed marketDataFeed = new YahooMarketDataFeed();
{
    /**
     * Sets the historical daily data from Yahoo
     * From : DD , MM , YYYY
     * To   : DD , MM , YYYY
     * Yahoo provides DAILY, WEEKLY and MONTHLY
     */
    marketDataFeed.setYahooParameters("01", "01", "2008",
                                      "31", "12", "2010",
                                      marketDataFeed.YAHOO_DAILY);
}
```

Yahoo, historical DAILY data from 1/1/2008 to 31/12/2010.

```
private final TBGMarketDataFeed marketDataFeed = new TBGMarketDataFeed();
{
    marketDataFeed.setMarketDataFeedId(112);
    marketDataFeed.setTbgMode(TBGMarketDataFeed.TBG_MODE_ONLINE);
    marketDataFeed.setTBGParameters("3", "8", "2011", "17", "9", "2012");
    // set the MarketDataEvent type, trades or candles ?
    //marketDataFeed.setMarketDataEvent(MarketDataEventType.TRADE_EVENT);
    marketDataFeed.setMarketDataEvent(MarketDataEventType.CANDLE_EVENT);
}
```

TheBonnotGang Feed, 1 minute candles from 3/8/2011 to 17/9/2012.

```
private final InteractiveBrokersAdapter interactiveBrokersAdapter = new InteractiveBrokersAdapter();
private final IBMarketDataFeed marketDataFeed = new IBMarketDataFeed(interactiveBrokersAdapter);
```

InteractiveBrokers Feed.

MarketDataEvent

A market data event can be categorized in two types: TickEvent and CandleEvent.

TickEvent

Contains the last Ask & Bid Price of a security and the last price which the security traded at.

- Symbol
- Price
- Size
- Volume
- Timestamp
- AskPrice
- AskSize
- BidPrice
- BidSize

CandleEvent

Contains the open, close, high and low price of a certain period of time.

- Symbol
- Timestamp
- OpenPrice
- HighPrice
- LowPrice
- ClosePrice
- Volume

Declaration

```
private final IMarketDataFeed marketDataFeed = new XXXMarketDataFeed();
{
    //marketDataFeed.setMarketDataEvent(MarketDataEventType.TRADE_EVENT);
    marketDataFeed.setMarketDataEvent(MarketDataEventType.CANDLE_EVENT);
}
```

MarketDataEvent should be setted in according with the MarketDataFeed implementation.

Strategy

The Strategy entity is one of the main components, it can be used through the TradingSystem entity. Even if it is possible to directly access to the Strategy component we suggest to use the TradingSystem component because it adds some useful utilities and hides the logical parts.

In the Strategy context four different events can happen:

- **onStart**
Event generated at the start of the strategy.
- **onStop**
Event generated at the stop of the strategy.
- **onEvent**
Event generated upon receipt of a market event (CandleEvent or TickEvent).
- **onError**
Event generated upon the occurrence of an error.

Declaration

```
public class EmptyStrategy implements IStrategy {  
  
    @Override  
    public void onStart() {  
    }  
  
    @Override  
    public void onStop() {  
    }  
  
    @Override  
    public void onEvent(Object event) {  
    }  
  
    @Override  
    public void onError(Messages msg) {  
    }  
}
```

This is an empty Strategy skeleton.

TradingSystem

TradingSystem is the main object to be used when you want to implement a new strategy, it contains a number of basic functions and a set of utility functions.

- **TradingSystemName**
- **TradingSystemDescription**
- **TradingSystemParameter**
Loads a parameter value from the external parameter configuration file.
- **getSecurityBySymbol**
Returns the security object from its symbol.
- **closeAllOpenPositions**
It closes all the open position in portfolio.
- **closePositionForSymbol**
It closes a position for a given symbol.
- **scheduleReportGeneration**
Permits to generate a report every X seconds.
- **getSharesFor**
Returns a number of shares for a given amount of money.
- **TrendTracker**
Keep track of the trend for a given symbol.
- **PositionTracker**
Keep track if a position is open in which side for a given symbol
- **Subscribe Securities**
Subscribe a security or a list of securities to the MarketDataFeed.

Declaration

```
public class EmptyTS extends TradingSystem {  
  
    public EmptyTS(){  
        setTradingSystemName("Empty TS");  
        setTradingSystemDescription("Empty Trade System");  
    }  
  
    public static void main(String[] args) {  
        new EmptyTS().start();  
    }  
}
```

This is a minimal empty TradingSystem.

How it all goes together

It's time to tie all together and build a working strategy skeleton.

First, we tie **TradingSystem** and **Strategy** together.

```
public class MySkelStrategy extends TradingSystem implements IStrategy {

    public MySkelStrategy(){
        setTradingSystemName("My Skeleton Strategy");
        setTradingSystemDescription("Use this to build something more unique!");
    }

    @Override
    public void onStart() {
        // On Strategy starts
    }

    @Override
    public void onStop() {
        // On Strategy stops
    }

    @Override
    public void onEvent(Object event) {
        // On MarketEvent
    }

    @Override
    public void onError(Messages msg) {
        // On Error occurs
    }

    public static void main(String[] args) {
        new MySkelStrategy().start();
    }
}
```

Then, we add the **Account**, the **Broker**, the **MarketDataFeed** and some **securities**.

We use the implementation of **PaperAccount**, **PaperBroker** and **BogusMarketDataFeed**.


```

public class MySkelStrategy extends TradingSystem implements IStrategy {

    private final PaperAccount account = new PaperAccount();
    private final IBroker broker = new PaperBroker(account);
    private final BogusMarketDataFeed marketDataFeed = new BogusMarketDataFeed();

    public MySkelStrategy(){
        setTradingSystemName("My Skeleton Strategy");
        setTradingSystemDescription("Use this to build something more unique!");
        setBroker(broker);
        setMarketDataFeed(marketDataFeed);
        subscribeSecurities(new LoadSecurities(SecurityType.STK, "SMART",
                                                Currency.USD, "XOM,WMT,AAPL").getSecurities());
    }

    @Override
    public void onStart() {
        // On Strategy starts
    }

    @Override
    public void onStop() {
        // On Strategy stops
    }

    @Override
    public void onEvent(Object event) {
        // On MarketEvent
    }

    @Override
    public void onError(Messages msg) {
        // On Error occurs
    }

    public static void main(String[] args) {
        new MySkelStrategy().start();
    }
}

```

We added 3 securities: Exxon (XOM), WalMart (WMT) and Apple (AAPL).
 If we execute this strategy we obtain this output:

```

WARN - Cannot find config.proprieties file, proceeding without it.
INFO - Found 3 symbols!
INFO - Strategy My Skeleton Strategy started [Wed Jan 16 12:07:41 CET 2013]
INFO - Strategy Description: Use this strategy to build something more unique!
INFO - Using default storeService...DEFAULT_STORESERVICE
INFO - RunID: [UserID:tbg][RunID:NA][Description:][StrategyName:][StrategyClass:]
WARN - No MarketDataEvent specified! proceeding using default: TRADE_EVENT
INFO - Initializing engine URI 'CEPEngine' version 4.3.0
INFO - Connected to PaperBroker.
INFO - Connected to BogusMarketDataFeeder.
INFO - XOM subscribed.
INFO - WMT subscribed.
INFO - AAPL subscribed.
INFO - Active MarketDataFeed...

```

Everything is working properly but we cannot see any market data coming in, this is because we did not implemented the `onEvent()` method.

Let's do this:

```
@Override
public void onEvent(Object event) {
    // On MarketEvent
    Log.info(event);
}
```

... and this is the output :

```
WARN - Cannot find config.proprieties file, proceeding without it.
INFO - Found 3 symbols!
INFO - Strategy My Skeleton Strategy started [Wed Jan 16 12:07:41 CET 2013]
INFO - Strategy Description: Use this strategy to build something more unique!
INFO - Using default storeService...DEFAULT_STORESERVICE
INFO - RunID: [UserID:tbg][RunID:NA][Description:][StrategyName:][StrategyClass:]
WARN - No MarketDataEvent specified! proceeding using default: TRADE_EVENT
INFO - Initializing engine URI 'CEPEngine' version 4.3.0
INFO - Connected to PaperBroker.
INFO - Connected to BogusMarketDataFeeder.
INFO - XOM subscribed.
INFO - WMT subscribed.
INFO - AAPL subscribed.
INFO - Active MarketDataFeed...
INFO - Time: Wed Jan 16 12:16:51 CET 2013 Symbol: AAPL Price: 316.9450596285343 Size: 1.0 Volume:
40 AskPrice: 316.87728439677926 BidPrice: 316.99397713513684 askSize: 46.0 bidSize: 17.0
INFO - Time: Wed Jan 16 12:16:51 CET 2013 Symbol: AAPL Price: 317.0716115562461 Size: 1.0 Volume:
26 AskPrice: 317.09253483060394 BidPrice: 317.1575639435422 askSize: 8.0 bidSize: 86.0
INFO - Time: Wed Jan 16 12:16:51 CET 2013 Symbol: WMT Price: 251.992572631073 Size: 1.0 Volume:
876 AskPrice: 251.98062041540038 BidPrice: 252.05596636994105 askSize: 15.0 bidSize: 11.0
INFO - Time: Wed Jan 16 12:16:51 CET 2013 Symbol: AAPL Price: 317.11020039359397 Size: 1.0
Volume: 738 AskPrice: 317.0997902705789 BidPrice: 317.1181056974891 askSize: 91.0 bidSize: 94.0
INFO - Time: Wed Jan 16 12:16:51 CET 2013 Symbol: AAPL Price: 317.02883432536305 Size: 1.0
Volume: 587 AskPrice: 316.99733044360397 BidPrice: 316.91270027859775 askSize: 19.0 bidSize: 62.0
[...]
[...]
```

Store Service

The storeService handles the storing of the persistence data like Account information, history of Orders, Executions, Open & Close Positions, etc...

By default the tbq-Quant Strategy is using an embedded database called H2DB, is anyway possible to use a custom interface to MySQL, Oracle, MSSQL, etc...

The use of a persistence layer is not required in order to execute a strategy but is strongly recommended.

Features:

- StoreAccount
- StorePosition
- StoreHistoricalPosition
- StoreExecution
- StoreOrder
- StoreSecurity
- StoreMap

Intraday Data

Note that even if you can use the storeService to save market quotes this is not an advisable option because relational databases are not suitable for storage of huge time series. We will provide a dedicated service to storing market quotes.

Implementations

- **MockStoreService**
Specify to use this storeService implementation if you do not want to use a storeService. *We strongly suggest not to use this implementation.*
- **H2DBStoreService**
This is the default implementation, you do not need to specify it.
- **MySQLStoreService** (* not available in beta release)
Optional service uses an external MySQL DB.
- **CSVStoreService**
Saves to CSV file format, very basic implementation.
We strongly suggest not to use this implementation.

Report Service

This is an highly customizable component for producing reports.

`executeReport()` is the main method to implement.

Implementations

- ***TextReportService***

Produces very simple report in 3 way:

- Output only
- Save to File only
- Output & Save to File

Declaration

```
private final IReportService reportService = new TextReportService();
{
    reportService.setReportType(ReportType.OUTPUT_ONLY);
}
```

```
private final IReportService reportService = new TextReportService("C:\\test\\");
{
    reportService.setReportType(ReportType.OUTPUT_AND_STORE);
}
```

It will show the output report and will write a txt file in C:\test\.

Alarm Service

Not yet implemented.

CEP Provider

Complex event processing (CEP) delivers high-speed processing of many events across all the layers of an organization, identifying the most meaningful events within the event cloud, analyzing their impact, and taking subsequent action in real time.

tbg-Quant supports CEP through 2 engines: Siddhi and Esper.

Implementations

- **PassThrough Provider**

Used by default it simply forward all the input events to the output without filtering.



- **Siddhi CEP engine** <http://wso2.com/products/complex-event-processor/>

Siddhi CEP is a lightweight, easy-to-use Open Source Complex Event Processing Engine (CEP) under Apache Software License v2.0. Siddhi CEP processes events which are triggered by various event sources and notifies appropriate complex events according to the user specified queries. Siddhi was started as a research project initiated at University of Moratuwa, Sri Lanka, and now being improved by WSO2 Inc.

tbg-Quant package embeds the Siddhi library.

- **Esper CEP engine** <http://www.espertech.com>

Esper is a component for complex event processing (CEP).

Esper and Event Processing Language (EPL) provide a highly scalable, memory-efficient, in-memory computing, SQL-standard, minimal latency, real-time Big Data processing engine for medium to high-velocity and high-variety data.

The Event Processing Language (EPL) is a declarative language for dealing with high frequency time-based event data.

Esper library is not embedded in the tbg-Quant package, you need to download the esper-X.y.z.jar library and add it to the tbg-Quant library (lib/) folder.

Declaration

Siddhi

```
/**
 * Sets CEP Provider, using SIDDHI
 */
private ICEPPProvider CEP = new SiddhiCEPProvider(this);
{
    final String [] QUERY = {
        "from candleStream [win.time(100)] ",
        "insert into candle symbol, closePrice, avg(closePrice) as avgClose, volume as vol",
        "group by symbol"
    };
    CEP.setCepQuery(QUERY);
}

/**
 * Costructor
 */
public HelloSiddhi() {
    [...]
    setCEPProvider(CEP);
}
```

Esper

```
/**
 * Sets CEP Provider, using ESPER
 */
private ICEPPProvider CEP = new EsperCEPProvider(this);
{
    final String [] QUERY = {
        "select symbol, price, avg(price) from tick.win:time_batch(5 sec)"
    };
    CEP.setCepQuery(QUERY);
}

/**
 * Costructor
 */
public HelloEsper() {
    [...]
    setCEPProvider(CEP);
}
```

For support about the EPL (Event Processing Languages) please refer to the Siddhi/Esper website.

Conclusion

In this chapter we saw the tbq-Quant model and the main components.

We went through the Account and Broker components, we talked about the MarketDataFeed and the Strategy and TradingSystem components.

We finally built a working Skeleton Strategy using the bogus market simulator (BogusMarketDataFeed) and the paper account and broker.

Now, you should be able to use this Skeleton code in order to build something more unique.

You can change the MarketDataFeed from Bogus to Yahoo for using daily or weekly data, or TBGMarketDataFeed for using 1 minute candles.

You can add some trading logic inside the `onEvent()` method, send orders to the PaperBroker and assessing the performances by checking the reports generated with the ReportService component.

In the next chapter "In Action", many aspects will be studied in more detail. We'll see how to write a simple moving averages crossover strategy.

In Action

After reading this chapter, you should be able to understand how to create a new strategy, how to test it and finally how to trade(execute) it.

Create, Test, Trade !

Create a Strategy

To create or edit a strategy you can choose between using the tbg-QuantDesk tool or your favourite IDE (like Eclipse, Net Beans, etc.).

Warning

Explaining how to use and setup an IDE is not the goal of this chapter.

We are planning to release the free version of tbg-QuantDesk for mid-2013.

Pre-Requisites

You can find a zipped package inside the folder examples-src within the tbg-Quant root. Or download the last **tbg-Quant-vXXX-examples-src** from the website.

Once you have the examples-src package, unzip it and import it in your favourite IDE.

Please follow the instructions below:

Eclipse IDE

<http://help.eclipse.org/helios/index.jsp?topic=%2Forg.eclipse.platform.doc.user%2Ftasks%2Ftasks-importproject.htm>

Netbeans IDE

<http://netbeans.org/kb/docs/java/project-setup.html#existing-java-sources>

Add the libraries

Once you imported the project into your IDE workspace, in order to compile it you have to add the libraries contained in the **lib** and **tbg** folders inside the tbg-Quant root.

Without this step you won't be able to compile any strategy.

Create your first Strategy

The simpler way is to clone an already existing strategy.

For this purpose in the examples package you can find the **Skel** Strategy example.

Use it like a starting point to code something more unique.

A Typical Strategy Structure

Let's break down the Strategy main structure:

Declaration head

```
public class Skel extends TradingSystem implements IStrategy{
```

All the strategies must extend the TradingSystem class which implements the base strategy features.

Components declaration

```
private final PaperAccount account = new PaperAccount(20000, Currency.USD);  
private final IBroker broker = new PaperBroker(account);  
private final IMarketDataFeed marketDataFeed = new BogusMarketDataFeed();
```

This section declares which component is going to be used: the account, the broker and the MarketDataFeed. We specify that our initial capital is \$20000.

Setting up

```
public Skel() {  
    setTradingSystemName("Skel");  
    setTradingSystemDescription("empty description");  
    setBroker(broker);  
    setMarketDataFeed(marketDataFeed);  
  
    subscribeSecurities(new LoadSecurities(SecurityType.STK,"SMART",  
                                           Currency.USD, "IBM,XOM,AAPL,GOOG").getSecurities());  
}
```

This section is used to wire the components together.

TradingSystemName, Description, Broker, MarketDataFeed and Securities.

Actions start, stop, error, event

```
@Override
public void onStart() {
    Log.info("START");
}
```

What to do when the strategy starts ?

```
@Override
public void onStop() {
    Log.info("STOP");
}
```

What to do when the strategy stops ?

```
@Override
public void onError(Messages msg) {
    // TODO Auto-generated method stub
}
```

What to do when some error occurs ?

```
@Override
public void onEvent(Object event) {
    Log.info(event);
}
```

What to do when an event (trade or candle) occurs ?

This is the most important method, when we receive a trade event, what are we going to do with it ?

onEvent() is the place where the strategy code is.

Starter

```
public static void main(String[] args) {
    new Skel().start();
}
```

In Action: SimpleMovingAverage crossover

Let's suppose we need to implement two SMA cross strategy system using daily data from Yahoo Finance.

Once we cloned the Skel TradingSystem we first need to change the MarketDataFeed from Bogus to Yahoo.

Set MarketDataFeed

Replace

```
private final IMarketDataFeed marketDataFeed = new BogusMarketDataFeed();
```

with YahooMarketDataFeed and specify the period and the timeframe:

```
private final YahooMarketDataFeed marketDataFeed = new YahooMarketDataFeed();
{
    /**
     * Sets the historical daily data from Yahoo
     * From : DD , MM , YYYY
     * To   : DD , MM , YYYY
     * Yahoo provides DAILY, WEEKLY and MONTHLY
     */
    marketDataFeed.setYahooParameters("01", "01", "2008",
                                     "31", "12", "2010",
                                     marketDataFeed.YAHOO_DAILY);
}
```

At this point you should be able to run the strategy and get real historical data from Yahoo.

Set Securities

We target IBM & Apple Inc (which symbol is AAPL)

```
subscribeSecurities(new LoadSecurities(SecurityType.STK, "SMART", Currency.USD,
                                       "IBM,AAPL" ).getSecurities());
```

Strategy Core Logic

This is where we need to code our system strategy.

```
@Override
public void onEvent(Object event) {
    Log.info(event);
}
```

First we declare our two moving average, 60 & 100 days:

```
SMA sma1 = new SMA(60);
SMA sma2 = new SMA(100);
@Override
public void onEvent(Object event) {
    Log.info(event);
}
```

...then

we set inside the constructor the `minimumPeriods` equal to the period of the slow sma.

We cast the event to the `CandleEvent` because Yahoo provides daily candles.

We `add the closeprice` to our two moving averages.

```
SMA sma1 = new SMA(60);
SMA sma2 = new SMA(100);
@Override
public void onEvent(Object event) {
    //log.info(event);

    CandleEvent ce = (CandleEvent) event;
    String symbol = ce.getSymbol();

    sma1.add(symbol, ce.getClosePrice());
    sma2.add(symbol, ce.getClosePrice());

    // be sure we reached the minimun events
    if (getSystemActivation(symbol)){
        // Buy & Sell logic here !!
    }
}
```

Inside the `getSystemActivation()` block we write our Buy&Sell logic.

Buy & Sell Logic:

BUY If position for symbol X does not exist and $sma1 > sma2$

SELL if position for symbol X exists and $sma1 < sma2$

```
if (sma1.getValue(symbol) > sma2.getValue(symbol))
    trendTracker.setTrendSide(symbol, TrendSide.UP);
else
    if (sma1.getValue(symbol) < sma2.getValue(symbol))
        trendTracker.setTrendSide(symbol, TrendSide.DOWN);
    else
        trendTracker.setTrendSide(symbol, TrendSide.FLAT);
```

And here is the part where we decide to send the signal.

```

if (trendTracker.getTrendSide(symbol)==TrendSide.UP){
    if (positionTracker.getStatusForSymbol(symbol)==PositionTracker.NON_EXISTENT){
        Log.info(ce.getTimestamp().getDate() +
            " BUY "+symbol+" @" +ce.getClosePrice());
    }else{
        Log.info(ce.getTimestamp().getDate() +
            " SELL "+symbol+" @" +ce.getClosePrice());
    }
}
}

```

We write a log message with the signal BUY or SELL.

This is the output for this strategy ran with Yahoo daily data on IBM & AAPL between 1/1/2008 & 31/12/2010:

```

WARN - Cannot find config.properties file, proceeding without it.
INFO - Found 2 symbols!
INFO - Strategy Tutorial1 started [Mon Jan 14 18:02:49 CET 2013]
INFO - Strategy Description: empty description
INFO - START
INFO - Using default storeService...DEFAULT_STORESERVICE
INFO - RunID: [UserID:tbg][RunID:NA][Description:][StrategyName:][StrategyClass:]
INFO - MarketDataEventType setted to CANDLE_EVENT
INFO - Initializing engine URI 'CEPEngine' version 4.3.0
INFO - Connected to PaperBroker.
INFO - Connected to YahooMarketDataFeed.
INFO - Fetching IBM from Yahoo...
INFO - Fetching AAPL from Yahoo...
INFO - Active MarketDataFeed...
INFO - Tue May 27 00:00:00 CEST 2008 BUY AAPL @186.43
INFO - Mon Mar 31 00:00:00 CEST 2008 BUY IBM @115.14
INFO - Mon Aug 25 00:00:00 CEST 2008 SELL AAPL @172.55
INFO - Fri Mar 27 00:00:00 CET 2009 BUY AAPL @106.85
INFO - Wed Sep 03 00:00:00 CEST 2008 SELL IBM @118.34
INFO - Thu Feb 26 00:00:00 CET 2009 BUY IBM @88.97
INFO - Tue Sep 14 00:00:00 CEST 2010 SELL AAPL @268.06
INFO - Tue Sep 21 00:00:00 CEST 2010 BUY AAPL @283.77
INFO - Mon Mar 29 00:00:00 CEST 2010 SELL IBM @128.59
INFO - Tue May 11 00:00:00 CEST 2010 BUY IBM @126.89
INFO - Thu Jul 08 00:00:00 CEST 2010 SELL IBM @127.97
INFO - Wed Sep 01 00:00:00 CEST 2010 BUY IBM @125.77
INFO - Sending INVOKE_STOP...
INFO - STOP
INFO - Connection to YahooMarketDataFeed closed.
INFO - Connection to PaperBroker closed.
INFO - Elaborated in 4480.0 millis.
INFO - H2 Connection safely Closed.
INFO - STRATEGY STOPPED.

```

Now it is time to send orders to the Broker (PaperBroker) and see if this strategy is profitable.

Execute Orders

Create an order and invest 5000 USD

```
Order order = new Order();
order.setSecurity(getSecurityBySymbol(symbol));
order.setOrderSide(OrderSide.BUY);
order.setOrderType(OrderType.MARKET);
order.setQuantity(getSharesFor(5000, ce.getClosePrice()));
broker.sendOrder(order);
```

This code will send to the broker a BUY order at MARKET for the equivalent of 5000 USD shares of 'symbol'.

To close an open position you can both insert an opposite order or use the following command:

```
closePositionFor(symbol);
```

onStop()

By adding the command `closeAllOpenPosition()` you force the Strategy to close all the opened positions before stop.

```
public void onStop(){  
    Log.info("onStop(): ");  
    closeAllOpenPositions();  
}
```

This is the output:

```
INFO - Sending orders to close open position...  
INFO - EVENT: Fri Dec 31 00:00:00 CET 2010 FILLED ORDER 13 qta: 39.0 Action: SELL Filled: 39  
@Price:146.76 symbol: IBM  
INFO - EVENT: Fri Dec 31 00:00:00 CET 2010 FILLED ORDER 14 qta: 17.0 Action: SELL Filled: 17  
@Price:322.56 symbol: AAPL  
INFO - Waiting positions to be closed...  
INFO - All positions have been closed.
```

What we did

- Cloned the Skel Strategy example
- Changed BogusMarketDataFeed to YahooMarketDataFeed (Daily data from 1/1/2008 to 31/12/2010)
- Changed Securities to IBM & AAPL (Apple)
- Wrote the Buy&Sell logic based on two moving averages cross
- Sended the orders to the broker (PaperBaroker)
- Added the instruction to close the open positions before stop the strategy

You should now able to write your own simple Strategy, for questions, support and help please contact us or post on the community forum.

Test a Strategy

In order to test a strategy you should need to get some basic statistics. This is done by using the **ReportService** service.

Let's add this service on the previous example.

In Action: Assessing SimpleMovingAverage crossover strategy

Add this lines just after the broker declaration

```
private final IReportService reportService = new TextReportService();
{
    reportService.setReportType(ReportType.OUTPUT_ONLY);
}
```

And inject the service :

```
public Tutorial1() {
    setTradingSystemName("Tutorial1");
    setTradingSystemDescription("empty description");
    setBroker(broker);
    setReportService(reportService);
    setMarketDataFeed(marketDataFeed);
    subscribeSecurities(new LoadSecurities(SecurityType.STK, "SMART",
        Currency.USD, "IBM, AAPL").getSecurities());
    setMinimunSystemPeriods(100);
}
}
```

And this line in the onStop() method:

```
@Override
public void onStop() {
    Log.info("STOP");
    closeAllOpenPositions();
    accountReport();
}
```

By calling the accountReport() is possible to print a set of basic statistics useful to asses our strategy.

Now execute the Strategy, a full set of basic statistics will be printed at the end, just like the following log:

Account Report

```
INFO - >----- ACCOUNT -----<
INFO - ACCOUNT_CODE:PaperAccount
INFO - ACCOUNT_TYPE:Standard
INFO - CURRENCY:USD
INFO - INITIAL_CAPITAL:20000.0
INFO - CASH_BALANCE:30447.86
INFO - INVESTED_CAPITAL:0.0
INFO - REALIZED_PNL:0.0
INFO - UNREALIZED_PNL:0.0
INFO - EQUITY_PNL:10447.86
INFO - >----- PORTFOLIO -----<
INFO - No position opened.
```

Total Statistics (Long + Short)

```
INFO - >----- STATS -----<
INFO - Initial Capital      :      20000.0
INFO - Ending Capital      :      30447.86
INFO - Total Profit & Loss :      10447.86 ( 52.23 % )
INFO - P&L Standard Dev.   :      2759.14
INFO - Annual Return       :      3482.62 ( 17.41 % )
INFO - Sharpe Ratio        :      1.43
INFO - Profit Factor       :      2.5
INFO - Total Trades        :      7
INFO - Winners             :      5 ( 71.42 % )
INFO - Losers              :      2 ( 28.57 % )
INFO - Equals              :      0 ( 0.0 % )
INFO - Open Trades         :      0
INFO - Average Trade       :      1492.55
INFO - Avg Winner Trade    :      2230.9
INFO - Avg Loser Trade     :      -353.34
INFO - Max Winner Trade    :      7415.66
INFO - Max Loser Trade     :      -356.46
```

Long Statistics

```
INFO - >----- LONG -----<
INFO - Long Profit & Loss      : 10447.86 ( 52.23 % )
INFO - Long P&L Standard Dev.  : 2759.14
INFO - Long Annual Return      : 3482.62 ( 17.41 % )
INFO - Long Sharpe Ratio       : 1.43
INFO - Long Profit Factor      : 2.5
INFO - Long Total Trades       : 7
INFO - Long Winners            : 5 ( 71.42 % )
INFO - Long Losers             : 2 ( 28.57 % )
INFO - Long Equals             : 0 ( 0.0 % )
INFO - Long Open Trades        : 0
INFO - Long Average Trade      : 1492.55
INFO - Long Avg Winner Trade    : 2230.9
INFO - Long Avg Loser Trade    : -353.34
INFO - Long Max Winner Trade   : 7415.66
INFO - Long Max Loser Trade    : -356.46
```

Short Statistics

```
INFO - >----- SHORT -----<
INFO - Short Profit & Loss      : 0.0 ( 0.0 % )
INFO - Short P&L Standard Dev.  : 0.0
INFO - Short Annual Return      : 0.0 ( 0.0 % )
INFO - Short Sharpe Ratio       : 0.0
INFO - Short Profit Factor      : 0.0
INFO - Short Total Trades       : 0
INFO - Short Winners            : 0 ( 0.0 % )
INFO - Short Losers             : 0 ( 0.0 % )
INFO - Short Equals             : 0 ( 0.0 % )
INFO - Short Open Trades        : 0
INFO - Short Average Trade      : 0.0
INFO - Short Avg Winner Trade    : 0.0
INFO - Short Avg Loser Trade    : 0.0
INFO - Short Max Winner Trade   : 0.0
INFO - Short Max Loser Trade    : 0.0
INFO - >-----<
```

This strategy makes around 52% on IBM & Apple, Long only, between 1/1/2008 and 31/12/2010 with an initial capital of \$20k and fixed \$5k per trade.

Setting Commission Costs

In order to assess the strategy in a realistic way, it is necessary to consider the costs of commission, this can be set by accessing the PaperBroker:

```
private final IBroker broker = new PaperBroker(account);
{
    broker.setBrokerCommissions(SecurityType.STK, 10.0);
}
```

Sets \$10.0 per execution.

Setting Slippage

Slippage is the difference between estimated transaction costs and the amount actually paid, it can be set by accessing the PaperBroker:

```
private final IBroker broker = new PaperBroker(account);
{
    broker.setSlippage(0.5);
}
```

Trade (execute) a Strategy

Switching from Testing to real Execution is quite simple.

Once you have validated your strategy, you can switch to “live mode” by replacing this components:

- Broker
- Account
- MarketDataFeed

Referring to the example saw before, we just need to switch from PaperBroker/Account to real Broker/Account and change the Yahoo data provider to a real time data provider.

Suppose we are using InteractiveBrokers:

```
private final IAccount account = new IBAccount();
private final InteractiveBrokersAdapter interactiveBrokersAdapter = new InteractiveBrokersAdapter();
private final IBroker broker = new IBBroker(account, interactiveBrokersAdapter);
private final IBMarketDataFeed marketDataFeed = new IBMarketDataFeed(interactiveBrokersAdapter);
```

Then, we should say to InteractiveBrokers to feed us with daily data or we can just collect intraday quotes through our CEP engine and filter out the daily value, this is an aspect we cannot cover in this paragraph.

Appendix

All source code can be found online at the www.thebonnotgang.com website.
For any doubts or support request please feel free to contact us both via community forums or by email at info@thebonnotgang.com

Skeleton Strategy Code

```
package com.tbq.TradingSystem.tutorial0;

import com.tbq.adapter.bogus.marketdatafeed.BogusMarketDataFeed;
import com.tbq.adapter.paper.account.PaperAccount;
import com.tbq.adapter.paper.broker.PaperBroker;
import com.tbq.core.model.broker.IBroker;
import com.tbq.core.model.strategy.IStrategy;
import com.tbq.core.model.types.Currency;
import com.tbq.core.model.types.Messages;
import com.tbq.core.model.types.SecurityType;
import com.tbq.strategy.TradingSystem;
import com.tbq.strategy.utils.LoadSecurities;

/**
 * Skeleton Strategy <br>
 */
public class MySkelStrategy extends TradingSystem implements IStrategy {

    private final PaperAccount account = new PaperAccount();
    private final IBroker broker = new PaperBroker(account);
    private final BogusMarketDataFeed marketDataFeed = new BogusMarketDataFeed();

    public MySkelStrategy(){
        setTradingSystemName("My Skeleton Strategy");
        setTradingSystemDescription("Use this strategy to build something more unique!");
        setBroker(broker);
        setMarketDataFeed(marketDataFeed);
        subscribeSecurities(new LoadSecurities(SecurityType.STK, "SMART", Currency.USD,
                                                "XOM,WMT,AAPL").getSecurities());
    }

    @Override
    public void onStart() {
        // On Strategy starts
    }

    @Override
    public void onStop() {
        // On Strategy stops
    }

    @Override
    public void onEvent(Object event) {
        // On MarketEvent
        Log.info(event);
    }

    @Override
    public void onError(Messages msg) {
        // On Error occurs
    }

    public static void main(String[] args) {
        new MySkelStrategy().start();
    }
}
```


Simple Moving Averages Crossover Strategy Code

```
/**
 * tbq-Quant examples package
 */
package com.tbq.TradingSystem.tutorial1;

import com.tbq.adapter.paper.account.PaperAccount;
import com.tbq.adapter.paper.broker.PaperBroker;
import com.tbq.adapter.yahoo.marketdatafeed.YahooMarketDataFeed;
import com.tbq.core.model.Order;
import com.tbq.core.model.broker.IBroker;
import com.tbq.core.model.report.IReportService;
import com.tbq.core.model.strategy.IStrategy;
import com.tbq.core.model.types.Currency;
import com.tbq.core.model.types.Messages;
import com.tbq.core.model.types.OrderSide;
import com.tbq.core.model.types.OrderType;
import com.tbq.core.model.types.ReportType;
import com.tbq.core.model.types.SecurityType;
import com.tbq.core.model.types.TrendSide;
import com.tbq.core.model.types.event.CandleEvent;
import com.tbq.indicator.SMA;
import com.tbq.service.report.TextReportService;
import com.tbq.strategy.TradingSystem;
import com.tbq.strategy.utils.LoadSecurities;

/**
 * Tutorial1 - Simple Moving Averages cross strategy <br>
 * <br>
 * Check out www.thebonnotgang.com online support.
 * <br>
 */
public class Tutorial1 extends TradingSystem implements IStrategy{

    // set the PaperAccount, $2000 initial capital
    private final PaperAccount account = new PaperAccount(20000, Currency.USD);
    private final IBroker broker = new PaperBroker(account);
    {
        broker.setBrokerCommissions(SecurityType.STK, 10.0);
        broker.setSlippage(0.5);
    }
    private final YahooMarketDataFeed marketDataFeed = new YahooMarketDataFeed();
    {
        /**
         * Sets the historical daily data from Yahoo
         * From : DD , MM , YYYY
         * To   : DD , MM , YYYY
         * Yahoo provides DAILY, WEEKLY and MONTHLY
         */
        marketDataFeed.setYahooParameters("01", "01", "2008",
                                           "31", "12", "2010",
                                           marketDataFeed.YAHOO_DAILY);
    }

    private final IReportService reportService = new TextReportService();
    {
        reportService.setReportType(ReportType.OUTPUT_ONLY);
    }

    public Tutorial1() {
        setTradingSystemName("Tutorial1");
        setTradingSystemDescription("Simple SMA cross");
        setBroker(broker);
        setReportService(reportService);
        setMarketDataFeed(marketDataFeed);
        subscribeSecurities(new LoadSecurities(SecurityType.STK, "SMART", Currency.USD,
                                                "IBM,AAPL").getSecurities());
        setMinimunSystemPeriods(100);
    }
}
```

```

@Override
public void onStart() {
    Log.info("START");
}

@Override
public void onStop() {
    Log.info("STOP");
    closeAllOpenPositions();
    accountReport();
}

SMA sma1 = new SMA(60);
SMA sma2 = new SMA(100);
@Override
public void onEvent(Object event) {
    //log.info(event);

    CandleEvent ce = (CandleEvent) event;
    String symbol = ce.getSymbol();

    sma1.add(symbol, ce.getClosePrice());
    sma2.add(symbol, ce.getClosePrice());

    // be sure we reached the minimun events
    if (getSystemActivation(symbol)){
        if (sma1.getValue(symbol)>sma2.getValue(symbol))
            trendTracker.setTrendSide(symbol,TrendSide.UP);
        else
            if (sma1.getValue(symbol)<sma2.getValue(symbol))
                trendTracker.setTrendSide(symbol,TrendSide.DOWN);
            else
                trendTracker.setTrendSide(symbol,TrendSide.FLAT);

        if (trendTracker.getTrendSide(symbol)==TrendSide.UP){
            if (positionTracker.getStatusForSymbol(symbol)==PositionTracker.NON_EXISTENT){
                // Open position for symbol
                Order order = new Order();
                order.setSecurity(getSecurityBySymbol(symbol));
                order.setOrderSide(OrderSide.BUY);
                order.setOrderType(OrderType.MARKET);
                order.setQuantity(getSharesFor(5000, ce.getClosePrice()));

                broker.sendOrder(order);
            }
        }else{
            if (positionTracker.getStatusForSymbol(symbol)==PositionTracker.POSITION_EXISTENT){
                closePositionFor(symbol);
            }
        }
    }
    // end system activated
}

@Override
public void onError(Messages msg) {
    // TODO Auto-generated method stub
}

public static void main(String[] args) {
    new Tutorial1().start();
}
}

```