Czech Technical University in Prague
Faculty of Information Technology
Department of Theoretical Computer Science

Bachelor's thesis

Robots for Algorithmic Trading on Foreign Exchange Market

Juraj Korček

Supervisor: Ing. František Štampach

12th May 2015

Acknowledgements

I would like to thank my supervisor František Štampach, who has been patient enough, for assisting me with the thesis. I would also like to thank Jakub Žitný for helping with measurement execution and Juraj Juráška and Martin Klepáč for their review and general advice.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on 12th May 2015 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Korček, Juraj. *Robots for Algorithmic Trading on Foreign Exchange Market.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2015.

Abstrakt

Táto bakalárska práca slúži ako úvod do sveta algoritmického obchodovania pre ľudí, ktorí majú predošlé skúsenosti s programovaním. Pre uvedenie čitateľa do problematiky obchodovania na burze menových párov bakalárska práca popisuje proces výberu makléra z pohľadu kritických parametrov ovplyvňujúcich obchodovanie. Ďalej, práca sa zaoberá tvorbou obchodovacích robotov uvedením najdôležitejších vlastností programovacieho jazyka MQL4, analýzou existujúcich, verejne dostupných robotov a implementáciou vlastného robota, ktorý môže slúžiť ako vzor pre ďalší vývoj. Naviac, práca sa snaží ohodnotiť výkon obchodovacích robotov s použitím trénerovania a testovania modelu, pričom samotné testovanie modelu nie je súčasťou existujúceho riešenia MetaTrader4.

Kľúčové slová burza menových párov, forex, algoritmické obchodovanie, učenie modelu, Expert Advisor, MetaTrader4, MQL4, výber makléra

Abstract

This thesis serves as an introduction into the world of algorithmic trading for people who possess some programming experience. In order to extend the reader's knowledge in terms of foreign exchnage trading, the thesis thoroughly

describes the process of broker selection with focus on the most important trade-affecting parameters. Furthermore, the thesis discusses a proper creation of trading robots by introducing the most important features of MQL4 programming language, by analyzing publicly available robots and implementing a new, well-designed robot which may act as a template for further robot development. Additionally, the thesis shows how to evaluate performance of the trading robots using model training and testing, which is not implemented in the state-of-the-art MetaTrader4 platform.

Keywords foreign exchange, forex, algorithmic trading, model learning, Expert Advisor, MetaTrader4, MQL4, broker selection

Contents

List of Figures

List of Tables

Introduction

Forex or *forex trading*, once a playing field solely for huge investment companies, has become available to masses recently. This was allowed to happen mainly due to the emergence of trading platforms such as MetaTrader, which are publicly available to be downloaded on the Internet free of charge and are thus accessible to everybody.

Initially, forex trading consisted of people sitting in front of computers all day long watching progress of the market. However, people are quite prone to weariness, which results in difficulties when it comes to maintaining their focus. What is more, people are usually likely to suffer from psychological swings such as fear, greed or hope which might result in abandonment of their strategy. These reasons caused an increased demand for robots performing algorithmic trading.

Robots behave according to the set strategy on all possible occasions while processing numerous market indicators simultaneously. Furthermore, unlike human beings, robots never lose focus. Another advantage of the robots is the fact that they render the requirement of watching the market all day long obsolete. As a result, people who are unable to trade in a full-time manner are granted the opportunity to actively participate in forex trade as well. Consequently, forex has become a hobby or the second source of income for a considerable number of people.

The objective of this thesis is to describe the whole process of entering forex trade – starting from the decision of selecting the right broker, through the choice of trading platform, continuing with an analysis and comparison of some publicly available robots and strategies they implement, ending with a basic guide to design and implement completely new strategies.

More specifically, the thesis will show process to choose suitable broker. A comparison of trading platforms will follow. Then the focus will turn to the basics of robot programming. Next, two publicly available robots will be examined and compared performance-wise to the newly designed robot built to show the construction process.

Chapter 1

State-of-the-Art

1.1 Existing Implementations

While numerous implementations exist, their authors are not particularly willing to share them entirely. In other words, even if it occurs that the author shares his robot, it is typically just the software itself with a description of the input parameters. Therefore, the user does not really see how the robot works on the inside and his only option is to test various combinations of the input parameters until plausible results are obtained. This is generally a time-consuming and, in case of testing using real accounts, also an expensive process without any promise of success.

One can assume that if somebody develops a really successful robot, he is unlikely to share it freely. One of the reasons supporting this claim is the fact that the development of a profitable algorithm is a very time-consuming process. Another reason is the fact that a person or, for that matter, a company develops the robot with the vision of profit. The desired profit can be achieved either by investing your money using the robot or selling the robot itself. While the former requires some initial capital, the latter needs none and thus might be more appealing to developers. The outcome of the previous hypotheses is that the lower the price, the lower the quality of the algorithm. While it does not apply strictly to every robot, I believe that this is generally a valid principle, unless the author is some kind of an altruist. On the other hand, an expensive robot does not necessarily mean that it is of a high quality as several scams have already been reported.

With regard to large financial institutions, their expensive-to-develop high-quality algorithms grant them a competitive advantage and thus these algorithms are considered to be a trade secret. Therefore, they are neither shared free of charge nor for money.

The outcome of the previous statements is that one should not really expect a reasonably profitable robot to be obtainable on the Internet. Hence, in case one wants to invest in the foreign exchange, the following two optimal solutions

exist:

1. To invest using a financial institution – in this case the person is not really able to affect a fashion in which the company invests. On the other hand, it is usually quite likely that the investment will bear some amount of profit. One should also consider the fact that financial institutions generate much higher profit from the person's investment than the person himself does.

2. To choose a broker, a trading platform and to develop a robot from scratch – an interesting option for a person who has sufficient time, determination, willingness to learn and who possess some financial reserves. In order to build a robot (also known as an *Expert Advisor*), one needs to learn and understand thoroughly the way how forex works. Once this has been accomplished, one should have enough insight to determine how to invest and after all whether to invest or not. To put it differently, the result will be either a profitable robot or a rational decision not to invest in the foreign exchange trade at all. One should also take into account the volatility[1] of the foreign exchange and consider the invested money to be already lost. It is not advisable to invest money that is intended for basic needs such as living or food.

1.2 Brokers Analysis and Comparison

The Internet offers a wide variety of foreign exchange market brokers differing in many parameters such as type, account size, number of currency pairs offered for trading, minimum opening deposit, fees, maximum leverage, demo account availability, trustworthiness and customer service. Among all available options it is considerably difficult for a newcomer to the trading world to find a broker that fits the trader's requirements most. This is caused by the fact that different brokers are suitable for different trading techniques. However, in the beginning it is quite challenging for a person to determine which trading technique suits him the best.

In the following chapters I will discuss each parameter in terms of importance and amount of attention one should pay to it.

1.2.1 Broker Type

Essentially, forex brokers can be divided into two main categories with respect to what their forex quotations are based on – *Dealing Desk broker* and

[1] "Volatility is the degree to which the price of currency tends to fluctuate within a certain period of time. For instance, in an active global trading day (24 hours), the euro/dollar exchange rate may change its value 18,000 times 'flying' 100-200 pips in a matter of seconds if the market gets wind of a significant event." [20, p. 10]

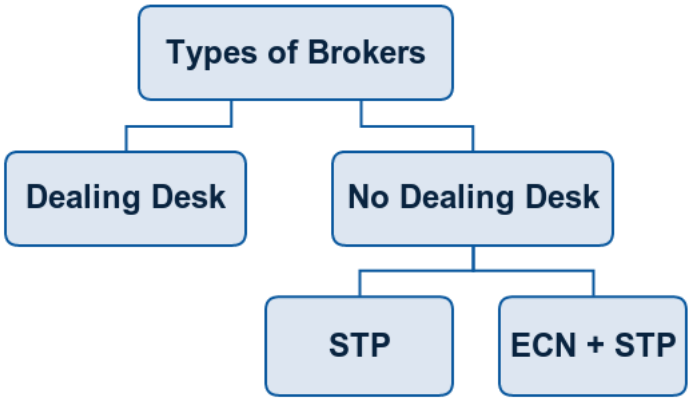

Figure 1.1: Forex broker types [6]

No Dealing Desk broker. Additionally, the No Dealing Desk broker has two subcategories – *Straight Through Processing* and *Electronic Communication Network*. See Figure 1.1.

Dealing Desk Broker

The Dealing Desk (DD) broker is also known as "market maker" because he literally creates the market for his clients by taking the other side of a client's trade. This has a few implications. As the broker himself controls the price, he is in a fairly low risk. Therefore, he can offer fixed *spreads*[2]. A fixed spread is advantageous for the trader as it is easier to calculate the *stop-loss*[3] and the *break-even points*[4] in advance. On the other hand, the broker's rates might not necessarily reflect the real interbank rates.

The DD broker makes money when the client loses the trade. Another source of his income stems from the spread.

No Dealing Desk Broker

As the name suggests, the orders in this case do not pass through the dealing desk. In other words, the broker does not take the other side of the client's trade. The No Dealing Desk (NDD) broker serves as a connecting element between the trader and the other side of the transaction.

[2] Spread is characterized as the difference between the bid and the ask price. [2, p. 20]

[3] "Stop Loss Order – Order type whereby an open position is automatically liquidated at a specific price. Often used to minimize exposure to losses if the market moves against an investor's position." [2, p. 50]

[4] "Breakeven Point (BEP) – In general, the point at which gains equal losses." [15]

Straight Through Processing NDD Broker

The Straight Through Processing (STP) means that the transactions are fully computerized and are immediately processed basically without any broker intervention by the other side of the transaction, which is, in case of the STP NDD broker, the broker's liquidity providers.

Unlike the spread of a DD broker, the spread with a NDD STP broker is usually variable. Consequently, it is considerably more difficult to calculate the exact stop-loss and break-even point. On the other hand, an advantage in comparison with a DD broker is the fact that the ask and the bid prices correlate with the real rates on the intrabank market.

The NDD STP broker does not make money from the difference between the bid and the ask price, as he just forwards the trades to his liquidity provider. The source of the income for the NDD STP broker is a small *markup*[5] on the spread.

In past, the usual way of trading was using a $T + N$ trading system. As everything was processed manually at that time, it usually took several days between the transaction date (the day the trade was initiated) and the settlement date (the date the deal was completed). The N denominated the maximum number of days for the trade to be completed. For example, $T + 3$ means that the deal at hand must be processed three days after the transaction initiation at the latest. So, in case the transaction is initiated on Monday, it must be settled by Thursday, supposing there are no public holidays in that time.

However, prices at the market are very volatile and thus can change considerably during a three-day period. Immediate processing offered by the STP significantly lowers the processing time and thus lowers the settlement risk – a risk that one of the parties will not be able to pay, resulting in the trade not being settled.

Electronic Communication Network NDD Broker

The Electronic Communication Network (ECN) broker is the most transparent broker type. The other side of the transaction is not taken by the broker's liquidity providers, as it is in the case of the STP NDD broker, but it is taken by different entities such as banks, hedge funds, mutual funds, other brokers or traders. However, bypassing a liquidity provider has several consequences.

A negative consequence is that without a liquidity provider it is almost impossible to trade small amounts of currency, since huge institutions usually trade in a large amount of currency and are not willing to take an offer for selling or buying small amounts, such as $100. Consequently, the ECN brokers

[5] "Markup – The difference between an investment's lowest current offering price among dealers and the higher price a dealer charges a customer." [15]

demand substantially high deposits when creating an account. The minimum deposit with a usual ECN broker is between $50 000 and $100 000

The positive consequence is that the trading is very transparent because one knows who is taking the other side of transaction. Rates are naturally the same as at the market because trading is taking place directly at the market. Due to the ECN NDD transparent nature, information regarding the *depth of market*[6] is also available, which is crucial for several trading strategies.

The spreads of the ECN broker are always variable. The ECN broker makes profit from *commissions*[7] charged at every trade. Some ECN brokers might also add a tiny markup to the spread.

Hybrid Broker

Many brokers who claim that they operate as a NDD STP broker actually represent a hybrid type between a DD and an NDD STP. They behave as a DD broker when the trader makes a losing trade. In other words, they take the other side of the trade when it is the losing trade for the trader, i.e. it is a winning trade for the broker. On the other hand, in case the trader executes a winning trade, brokers route it to the intrabank market (particularly their liquidity providers), which corresponds to a usual NDD STP broker behavior.

Analogically, the profit source of this broker type is the same as the DD broker's profit source when the trader loses his trade – meaning that the broker takes the other side of the trade and wins. When the trader wins his trade, the transaction is directed to the broker's liquidity providers and the broker himself makes money on a little spread markup.

1.2.2 Account Type

Foreign exchange market accounts are divided into three main categories based on the size of the minimum amount traded. The basic and, at the same time, the oldest type is a *standard account*, where the minimum allowed amount of currency is one *lot*. One lot is equivalent to 100 000 units of currency. So, for instance, when the trader wants to buy $100 000 , he purchases one lot of USD. Similarly, if the trader wants to buy 10 000 000 €, he purchases 100 lots of €.

However, 100 000 units of currency, which represents the smallest tradable amount, is a considerable amount of money for an average person. Therefore, in order to make the forex trading accessible to more people, *mini accounts* have been introduced. For traders with mini accounts the lowest tradable

[6] "Depth of Market (DOM) – A measure of the number of open buy and sell orders for a security or currency at different prices. The depth of market measure provides an indication of the liquidity and depth for that security or currency. The higher the number of buy and sell orders at each price, the higher the depth of the market." [15]

[7] "Commission – A transaction fee charged by a broker." [2, p. 44]

volume of currency is a *mini-lot*. A mini-lot is the equivalent of 10 000 units of currency.

Eventually, *micro accounts* have become available. Micro accounts allow traders to trade with a *micro-lot* as the smallest amount of currency. A micro-lot is the equivalent of 1000 currency units. \$1000 or 1000 € is still considered to be a substantial amount of money for an average person, however, with the help of leverage (see chapter 1.2.3) it is possible to trade 1000 units with as little as 10 real units in the trader's account. This has made trading on the foreign exchange market accessible to a wider public.

Recently, some brokers allowed trading with an amount of currency as small as one unit. It means, for instance, that a trader can open a position[8] with only \$1. There are not many brokers that offer this option. One of them is Oanda. However, it is not without limitations. In case of Oanda it is possible to trade such small amounts only manually, as the algorithmic software trading is not compatible with such uncommon lot sizes.

1.2.3 Maximum Leverage

Maximum leverage values range from 1:50 to 1:400 with the most usual rate being 1:100. Leverage is a tool to leverage the trader's capital. The higher the leverage, the higher the "virtual" capital available. However, it is considered to be a double-edged sword, as approaching the maximum leverage results in a higher potential profit, but also a higher possible loss. The following example will attempt to explain this concept.

A trader has \$5000 in his forex account with a leverage of 1:200. This means he can open a trading position worth up to \$1 000 000 ($5000 * 200$). If USD appreciates by 1%, the trader will make \$10 000 with the investment of only \$5000 , thus he will make a 200% profit.

However, a USD depreciation by 0.5% is sufficient for the trader to empty his account completely. To put it into perspective, the depreciation of USD by 0.5% in the EUR/USD currency pair means a change in exchange rate from 1.1477 to 1.1534. Considering that an average daily volatility for the currency pair EUR/USD in the last three months (as of January 2015) was approximately 90 *pips*[9] – if the rate changes at least six times the average of the average daily volatility, the trader can lose all his account ($6 \times 0.009 < 0.0057 = 1.1534 - 1.1477$).

[8] "Position – The amount of a security either owned or borrowed by an individual or by a dealer. In other words, it's a trade an investor currently holds open." [15]

[9] "Price Interest point (Pip) is the term used in currency market to represent the smallest price increment in a currency. It is often referred to as ticks or points in the market. In EUR/USD, a movement from .9018 to .9019 is one pip. In USD/JPY, a movement from 128.50 to 128.51 is one pip." [2, p. 19]

1.2.4 Margin, Margin Call and Stopout Level

If USD in the previous example depreciated by more than 0.5%, not only would it wipe out the trader's account, but also cause the account to go into negative numbers, meaning it would cause a loss to the broker as well. This loss would have to be exacted from the trader afterwards, which would be a very lengthy process.

To avoid this kind of losses the broker has a set of useful tools at his disposal. Although these tools are mainly to avoid the broker's losses, they also help the trader, as they notify him when his equity is dropping below appropriate levels.

Margin is the amount of money that serves as a kind of a "safety cushion". It can be considered as a deposit for opening the position. Additionally, it can be viewed as a limit for the trader's equity when an action is necessary.

The size of the margin is calculated as a fraction of the value of the position that is about to be opened. Usually, this fraction is derived from the maximum leverage value. So, if the maximum leverage is 1:100, the margin is 1% (1/100) of the value of the opened position. Analogically, in case the maximum leverage is 1:200, then the margin is 0.5% (1/200), and similarly with any value of the maximum leverage.

In order to be able to define the margin call, the stopout level and to provide an example we need to define further terms, such as balance, equity, used margin, available margin and margin level.

Balance is the amount of money in the account before opening a position. It changes only after the closing of a position, a withdrawal or depositing of funds.

Equity is the actual amount of money including the profits or the losses from the current open positions. The equity equals the balance if no positions are open.

Used margin is defined as the sum of the margins from all open positions.

Available margin is calculated as the difference between the equity and the used margin. It represents an amount of equity which is available for opening new positions or for covering the losses from the already open positions.

Margin level is defined as the equity over the used margin as percentage. It is a proportional representation of the equity available to be used for opening new positions or for covering the losses from the open positions.

Margin call is a warning issued by the broker for the trader informing that the trader's equity is at the same level as the used margin or lower. In other words, a margin call is issued when the margin level is equal or lower than 100%. It means that the trader is not allowed to open new positions, as he has no available margin to do so. The trader has three different options how to cope with a margin call. He can:

- put additional funds into the account. However, the bank transfer usu-

Table 1.1: Initial financial standing

Balance	**Equity**	**Used margin**	**Available margin**	**Margin level**
$6000	$6000	$4000	$2000	150%

Table 1.2: Financial standing after currency appreciation

Balance	**Equity**	**Used margin**	**Available margin**	**Margin level**
$6000	$6200	$4000	$2200	155%

ally has a certain processing time. This measure does not have an immediate effect and in case it is not executed in a timely manner, it will be ineffective.

- close the position to avoid additional loses
- ignore the warning with a hope that the market direction will change

Stopout level is a margin level limit at which the broker automatically closes the trader's open positions in order to avoid additional losses on the trader's side as well as possible losses on the broker's side. It can range from about 25% to 100%. If the stopout level is set to 100% it equals margin call. To put it differently, as long as stopout level is set to 100%, all the trader's positions will be closed immediately at the margin call.

The following example will attempt to elaborate on the above-mentioned concepts.

A trader opens a position worth $200 000 using his account with an actual balance of $6000 , a maximum leverage of 1:50 and a stopout level of 50%. The margin in this case will be $4000 ($200 000 $\times 1/50$). The trader's financial standing in the beginning is depicted in Table 1.1.

Let's say that the currency the trader had bought appreciated by 0.1%. Now, his position is worth $200 200 ($200 000 $\times 1.001$). The updated trader's financial standing is displayed in Table 1.2.

As the trader still has an available margin, he is allowed to open additional positions. Now, the trader's currency rapidly depreciates by 1.1%. His position is now worth $198 000 ($200 200 $\times (1 - 0.011)$). The trader's financial standing is portrayed in Table 1.3.

At this point, the trader has no available margin and therefore he is not allowed to open new positions. Moreover, the broker issues a margin call. If the broker's reaction to the margin call is the decision to close his position, his financial standing after the trade will correspond to Table 1.4. The trader loses $2000 in this trade.

Table 1.3: Financial standing after currency depreciation

Balance	**Equity**	**Used margin**	**Available margin**	**Margin level**
$6000	$4000	$4000	$0	100%

Table 1.4: Financial standing after closing the position

Balance	**Equity**	**Used margin**	**Available margin**	**Margin level**
$4000	$4000	$0	$4000	N/A

Table 1.5: Financial standing after ignoring margin call

Balance	**Equity**	**Used margin**	**Available margin**	**Margin level**
$6000	$2000	$4000	−$2000	50%

Assuming that the trader decides to ignore the margin call and the currency he had bought depreciates by another 1%, the value of his position reaches $\$196\,000\,(198\,000 \times (1 - 0.01))$ and the current financial standing is displayed in Table 1.5.

At this time the trader's equity is only 50% of his used margin. As the broker's stopout level is 50%, he automatically closes the trader's position. The final financial standing of the trader is depicted in Table 1.6. The trader loses $4000 in this trade.

To sum it up, the size of the margin and the stopout level are among the important factors that the trader should consider when choosing his broker. While in most cases the margin is correlated to the leverage level, some brokers offer different margins for different currency pairs regardless of the leverage. As a result, it is not recommended to ignore this parameter. As for the stopout level, if no statement exists about it on the broker's website, it is expected that it equals 100% (margin call). A margin call is usually not the only notification. Many brokers issue several additional notifications when the margin level is approaching the stopout limit.

Table 1.6: Final financial standing

Balance	**Equity**	**Used margin**	**Available margin**	**Margin level**
$2000	$2000	$0	$2000	N/A

1.2.5 Loss-limiting Mechanisms for the Trader

While in the previous chapter the broker's mechanisms for limiting the loss were discussed, now the mechanisms available to the trader will be analyzed. The loss-limiting mechanism for the trader based on choosing a suitable strategy or on abiding to some rules will not be discussed, as they are solely in the hands of the trader and do not influence the choice of the broker. The main focus will be aimed at the technical mechanisms offered by the broker.

The two significantly widespread tools are *stop-loss* and *trailing stop*. While the stop-loss is supported by all brokers, the trailing stop support is slightly lagging behind with some brokers (such as eToro).

The stop-loss is a very simple mechanism. The trader decides at which exchange rate his losses are too great to bear. If the exchange rate reaches this limit, the position is automatically closed. This is useful for traders who are not able to watch the market all day long. Thanks to the stop-loss this kind of traders will not lose all the capital in case of a steep market change in their absence.

The trailing stop works similarly to stop-loss. However, in case of the trailing stop the trader does not select the exact exchange rate, but a difference in pips, price or percentage instead. The advantage of the trailing stop is that it does not only protect against a loss, but it secures a profit as well. It is because the difference required for closing the position is calculated from the highest (or the lowest, depending on the position type) price since the introduction of the order. The following example will attempt to describe these concepts.

Trader A opens a EUR/USD long position[10] at 1.1362 and sets the stop-loss at 1.1342. At the exactly same time, trader B opens a EUR/USD long position at 1.1362 and sets the trailing stop at 20 pips (which corresponds to a decrease of 0.002 points).

Scenario 1: the exchange rate drops to 1.1310. However, the stop-loss of trader A was triggered at 1.1342, so his position was closed at this rate and thus his losses were minimized. Similarly, the position of trader B was closed after dropping by 20 pips (or 0.002 points), i.e. at the EUR/USD rate 1.1342 ($1.1362 - 0.002$). In this scenario, the result is exactly the same regardless of the mechanism being used.

Scenario 2: the exchange rate first rises to 1.1391 and then drops to 1.1310. For trader A nothing has changed. The stop-loss was triggered at 1.1342, his position was closed and the result is the same as in Scenario 1. However, for trader B the result is very different in this case, as he is using the trailing stop, which is triggered at the set difference from the peak. The peak in this scenario is 1.1391, so the stop was triggered and his position was closed at rate 1.1371 ($1.1391 - 0.002$). Trader B ended much better off than trader A.

[10] Long Position – An investment position that benefits from an increase in market price. When the base currency in the pair is bought, the position is said to be long. [2, p. 48]

In fact, trader B made a profit of 0.0009 per unit. It is thanks to the stop level being automatically adjusted when the market is moving in a profitable direction, hence the name trailing stop.

These two orders are quite common. However, many more exist. It is advisable to search through all options, find those that might suit the trader's needs and choose a broker based on this criterion if possible.

1.2.6 Minimum Opening Deposit

Minimum opening deposits range from $1 to $100 000. Usually, higher deposit levels are required by the brokers focused on corporate or institutional traders. Also, the deposits required by ECN brokers are approaching $100 000 because with an ECN broker the traders access the intrabank market where huge volumes of currency are traded, so it is not particularly feasible to trade small amounts.

1.2.7 Fees

For the broker, as for any privately owned company, the goal is a profit. One of the ways the broker is able to make money is by charging a fee. There are several types of fees the trader should be familiar with. Some of them depend on the broker type (DD, NDD STP, NDD ECN), others are more generic.

Spread

Although spread is not a fee itself, it can be considered a kind of a hidden fee. A spread is the difference between the ask and the bid price. Normally, it is decided automatically by the market. However, it is not the case with the Dealing Desk (DD) brokers (also known as Market Makers), as they are taking the other side of the transaction. Thus, they are creating the market and so deciding the amount of spread. In this way they are capable of increasing their profits and therefore decreasing the profits of the traders.

An example – if the ask price for the currency pair EUR/USD is 1.12089 and the bid price is 1.12039, then the spread has a value of 0.0005 (1.12089 − 1.12039) or 5 pips.

Markup on Spread

Putting a markup on a spread is a usual practice of the STP brokers and occasionally of the ECN brokers. The name exactly explains the operation. The broker takes the spread given to him by his liquidity provider (in case of STP) or the market (in case of ECN) and adds a small markup to it. It can vary from fractions of a pip (i.e. a change less than 0.0001 in one unit price for the EUR/USD pair) to several pips (a change by 0.0001 or more in one unit price for the EUR/USD pair). An example follows to describe this concept.

Let us assume that the broker receives an ask price of 1.12089 and a bid price of 1.12039 for the currency pair EUR/USD from his liquidity provider. Assume that the broker's policy is to put a one-pip markup on both the ask and the bid price. Therefore, what the trader sees is the ask price at 1.12099 and the bid price at 1.12029. The trader decides to buy 100 000 €. At the ask price of 1.12099 he has to pay the broker $112 099. The broker takes this amount and buys 100 000 € from his liquidity provider at the ask price of 1.12089, thus paying him $121 089. In the end, the trader pays $121 099 and gets his 100 000 €, the liquidity provider receives $121 089 and the broker earns $10 (121 099 − 121 089).

Any prospective trader should check with the broker what is the amount of markup that is put on top of the spread because the higher the markup, the harder it is for the trader to make a profit. However, not all brokers are willing to admit putting a markup on top of the spread. Therefore, one should trade with trustworthy brokers only. More on the topic of trustworthiness will be discussed in section 1.2.17.

Commission

Commission is a measure commonly employed by the ECN brokers. It is a small fee charged at the time of opening and closing of a position. It is typically set as a certain amount of dollars per one traded lot. Mostly, it fluctuates around $5 per one traded lot, i.e. $10 round trip ($5 for the position opening and $5 for the position closing). Occasionally, some brokers set the commission as a fixed value regardless of the traded amount.

An example – the trader using an ECN broker who charges a $5 commission (one-way) buys $10 000 which is 0.1 of a lot (one lot being 100 000 currency units). As a $5 commission is charged per one lot traded, the trader is charged $0.5 (10 000 /100 000 × 5). Then USD appreciates and the trader's position is now worth $10 010. The trader decides to close his position. He is charged again, this time $0.5005 (10 010/100 000 × 5). To sum up, the trader has made $8.9995 (10 010 − 10 000 − 0.5 − 0.5005) and the broker has made $1.0005 (0.5 + 0.5005).

Naturally, the broker charges a commission for the loss trades as well.

Rollover/Swap Charges

Rollover or swap charges are not fees in its essence, as the trader can both lose and profit on it. However, it is important to take them into account when building a strategy.

Swap charges are also known as overnight charges, as they are charged on all positions that stay open overnight when the exchange is closed. They are calculated from the difference between the annual interest rates of the currencies in the currency pair. The reason for this is that in order to open

the position, the trader is effectively borrowing one currency to buy another. For instance, if the trader wants to buy USD by trading the currency pair EUR/USD, he first borrows the necessary amount of EUR in order to buy the desired amount of USD. Naturally, an interest is paid on the borrowed money and collected on the invested money. Thus, in the above example the trader would pay an interest on the amount of EUR borrowed and at the same time he would collect an interest on the amount of USD owned. A rollover charge is the difference between the paid and the collected interest. In case the interest rate on the currency owned is higher than the interest rate on the currency borrowed, the trader collects a profit, otherwise he pays some loss.

The interest rate for every currency is set by the corresponding national bank. However, these interest rates are not exactly those which are taken into calculations. They are only the target interest rates that banks plan to meet within a certain fiscal period. Therefore, the actual interest rates used in the calculation of the rollover charges might differ to some extent. As these interest rates are annual, they need to be normalized to a one-day period.

The liquidity providers of the brokers usually apply a certain spread to the interest rates. Additionally, the brokers often put a certain markup either on top of the interest rate or on top of the final rollover amount. So, the rollover charges can be considered another source of income for the broker.

The trader does not need to calculate the rollover charges himself, as the broker does this for him automatically. A rollover charge is usually normalized per 100 000 units (i.e. one lot) and shown as an amount in the account currency – the currency that is put on the account when depositing. However, it is not always the case. Therefore, it is necessary to check how the broker calculates this figure. For example, some brokers show the rollover charge normalized to 10 000 units of the account currency instead of 100 000, while others show the charge as a number of pips of the traded currency pair. Yet again, an example will follow to clarify the above concepts.

The trader is trading the currency pair GBP/USD. The interest rate on GBP is 0.5% and 0.25% on USD. The exchange rate is 1.508. The trade's account currency is AUD, the GBP/AUD rate is 0.5272 and the AUD/USD rate is 1.258.

The trader decides to buy £200 000 That means he needs to borrow \$301 600 (200 000 $\times$1.508). He decides to leave his position open overnight. Therefore, the rollover charges are applied when the exchange market closes. For the sake of example simplicity, let's assume that the exchange rate did not move or that it was at the same level as when the position was opened. As the trader is holding GBP and the interest rate on GBP is higher than on USD, the trader will collect profit. Ideally, the amount of the earned money will be £1.3699 $((0.005 - 0.0025)/365 \times 200\,000)$.

The rollover charges per one unit of GBP/USD are 0.000006849 $((0.005 - 0.0025)/365)$, or 0.06849 pips. As it was mentioned before, the broker might display the rollover charge as an amount of the account currency per lot. In

this case it would be 1.299 AUD $((0.005 - 0.0025)/365 \times 100\,000 \times 1/0.5272)$. However, in reality, because of the broker's markup the trader's profit will be slightly lower.

In the same scenario, but the other way around, when the trader buys \$301 600 (worth £200 000), all the charges will be the same, only negative because the trader bought USD, which has a lower interest rate than the borrowed GBP:

- –£1.3699 $((0.0025 - 0.005)/365 \times 301\,600 \times 1.508^{-1})$
- –0.000006849 $((0.0025 - 0.005)/365)$ or –0.06849 pips per one unit of GBP/USD
- –0.8616 AUD $((0.0025 - 0.005)/365 \times 100\,000 \times 1.258)$ per one lot of USD, which is –1.299 AUD $(- - 0.8616 \times 1.508)$ per one lot of GBP

However, in reality, because of the broker's markup the trader's loss will be slightly higher.

It is important to note that because of the interest rate, the spread and the markup it might occur that (most often with currency pairs in which both currencies have a similar interest rate) the rollover charge, when both buying and selling, will be negative.

Another interesting thing is that on Wednesday the rollover charge is charged three times. The reason for this is the fact that the interests are calculated even for weekend days, but the foreign exchange trading does not take place on weekends, so it is necessary to compensate this. The same logic applies to banking holidays.

The forex exchange market is open without interruption from Sunday 21:00 GMT (Sydney exchange opening on Monday 8:00 AEDT – in summer) or 22:00 GMT (Sydney exchange opening on Monday 8:00 AEST – in winter) till Friday 21:00 GMT (New York exchange closing at 17:00 EDT – in summer) or 22:00 GMT (New York exchange closing at 17:00 EST – in winter). Each broker decides the time when he charges the rollover charges. Usually, it is in the evening, but the exact time may differ among brokers. For example, if the rollover time set by the broker is 17:00, then the position open at 16:59 will be subject to charges that day while the position open at 17:01 will not.

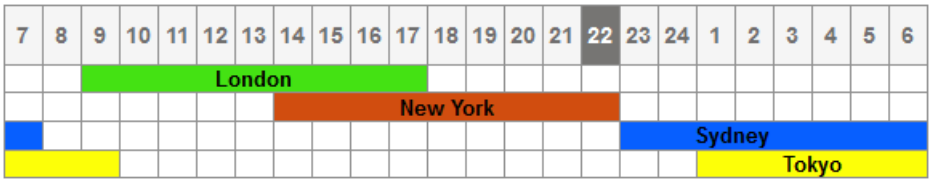

Figure 1.2: Market opening hours GMT+2 [14]

Withdrawal and Deposit

Brokers sometimes charge fees for a withdrawal and a deposit. These fees vary between different brokers and between different withdrawal and deposit options. Some options might be free of charge, while others not. Also, some brokers differentiate between local and international transactions – the international ones are charged with higher fees. It is also important to take into account that you might be charged twice for a single transaction – once by the broker and once by your financial institution.

Inactivity Fees

Most of the brokers charge inactivity fees when the trader does not open any position or does not execute any trade for a certain period of time. This period usually ranges between 30 and 90 days and the fee varies from $20 to $50.

Other Fees

Other fees may apply. It is recommended to study the broker's website and the conditions thoroughly so that the trader becomes informed about all his rights and duties. It is useful for minimizing unexpected losses or for detecting fraud.

1.2.8 Swapfree Account Availability

As the name suggests, a *swapfree account* is an account free of swap/rollover fees. Swap/rollover fees are described in detail in chapter 1.2.7. Swapfree accounts are also known as *Muslim accounts* or *Sharia Law accounts*, as they were primarily created to make trading available to Muslims, whose religion forbids profiting on interest rates. These accounts are offered only by a subset of brokers.

To register this kind of account, more often than not, the brokers require a piece of evidence that the trader is a Muslim. However, there are several ways how a non-Muslim trader can obtain this type of account:

- Choosing a broker that does not require the trader to be a Muslim in order to open the account. However, there are very few such brokers, so this makes the list of available brokers considerably shorter.
- Faking a piece of evidence of being a Muslim, as the broker is usually incapable of verifying the proof.
- In case the broker is located in one of the European Union member countries, the trader can contact the broker with a reference to the Charter of Fundamental Rights of the EU regarding religious discrimination [11, Ch. 21] and threaten him with a lawsuit. The broker will probably give

in and allow the trader to create the swapfree account regardless of his religion. Otherwise, the only way is to sue the broker.

Some brokers offer a swapfree account with exactly the same trading conditions as with a standard account. On the other hand, some brokers charge an additional fixed fee – usually a certain amount of USD per open position per night.

The brokers offering swapfree accounts often include in their Terms & Conditions that they reserve the right to close a swapfree account or transform it to a standard one without previous notice and without giving reasons for such an action. If the swapfree account is a part of the trader's long-term strategy, he should definitely examine the Terms & Conditions thoroughly.

As the trader can also profit from the swap/rollover charges, opening a swapfree account is not always the best solution. Whether the swapfree account is beneficial to the trader or not depends solely on his strategy.

1.2.9 Withdrawal and Deposit

As brokers offer a wide variety of deposit and withdrawal options, it is necessary to check if the broker supports those suitable for the trader. Another important thing to note is the fact that the deposit options offered by the broker might differ from the withdrawal options. Specifically, the withdrawal options are those that the trader should pay more attention to. Some brokers intentionally make the withdrawal more difficult, for instance by applying limits to the minimum amount of withdrawn money at once or by offering fewer withdrawal options compared to the deposit options.

This is a list of different withdrawal and deposit options:

- Bank wire
- Credit/debit card
- Electronic payment (PayPal, Skrill, WebMoney, etc.)
- Western Union
- Digital currency (Bitcoin)

Brokers usually apply certain deposit and withdrawal fees. These are described in chapter 1.2.7.

1.2.10 Number of Offered Currency Pairs

Forex brokers offer a wide range of currency pairs. The most traded currencies are called *Major currencies*. Currencies belonging into this group are USD, EUR, CHF, JPY, GBP, CAD, AUD and NZD. By combining these currencies

with USD seven *Major currency pairs* are obtained (EUR/USD, USD/JPY, GBP/USD, USD/CHF, AUD/USD, USD/CAD, NZD/USD).

Additionally, a combination of the Major currencies, except for USD, creates twenty-one *Cross currency pairs* (GBP/JPY, EUR/JPY, EUR/CHF or EUR/GBP, to name a few).

The advantage of the Major currency pairs is a high level of liquidity. In the forex market, liquidity pertains to a currency pair's ability to be bought and sold without causing a significant change in its exchange rate. A currency pair is said to have a high level of liquidity when it is easily bought or sold and there is a significant amount of trading activity for that pair [5]. Therefore, brokers can offer lower spreads so it is easier to profit on this type of currency pair.

Moreover, a combination of a Major currency with a non-Major currency, such as CZK, RUB, HUF, NOK, SEK, HKD, belongs to a so-called group of *Exotic currency pairs*. These currencies are usually traded neither very often nor in sizeable volumes. It means a lower level of liquidity because it is more difficult to execute a trade at a desired price. Therefore, brokers usually offer wider spreads for these currency pairs. However, the Exotic currencies might be more suitable than the Major currency pairs for some trading strategies.

A few brokers even offer combinations of two non-Major currencies, such as NOK/SEK. These kind of combinations are very uncommon and do not belong to any special group.

1.2.11 News Trading

News trading stands for trading during the time of significant economic announcements and therefore it is considered to be trading based on a fundamental analysis. The following list contains examples of such announcements:

- Interest rate decision
- Retail sales
- Inflation (consumer price or producer price)
- Unemployment
- Industrial production
- Business sentiment surveys
- Consumer confidence surveys
- Trade balance
- Manufacturing sector surveys

Every country has some more or less regular time frames for such announcements.

However, not all brokers allow the news trading. Those who do not allow the news trading usually freeze the market before the news announcement time and unfreeze it afterwards, so that it is impossible to open or close positions during that time. The reason for this is that during this time frame the market volatility is noticeably high, which usually causes higher spreads and possible significant slippage (see chapter 1.2.13). If the trader operates according to the news trading strategy, he should definitely verify whether the broker supports it or not.

1.2.12 Demo Account Availability

Demo accounts are priceless helpers for any novice trader who can try out different platforms, get accustomed to them, attempt different strategies, obtain valuable experience also by making mistakes. Furthermore, one may easily determine what type of trader he is, and all of that without spending a single cent. Many professionals advise not to start trading on a live account unless the trader is confident and profitable when trading on a demo account.

Basically, any decent broker should offer a demo account. If it is not the case, the trader should consider this as a warning sign. These accounts are for free. Some of them are time limited and their time range varies from two weeks to one month, while others are unlimited. An important thing to check is the extent to which the trading conditions for the demo accounts are the same when compared to the live accounts, as "most demo trading platforms are very similar to their live counterparts, but not exactly the same. There may be a difference in speed of execution, slippage, and platform reliability". [5]

1.2.13 Slippage

Slippage is an event when a market order is filled with price different from the one requested. This usually happens when the market is highly volatile or very illiquid. It can possibly have both a positive or a negative impact.

An example – the trader wants to trade the EUR/USD currency pair at a current exchange rate of 1.1286. He decides to buy 100 000 €, which means he needs to borrow and pay $112 860.

Scenario A: The market is highly volatile because of the news announcement, slippage occurs and the trader's order gets filled at 1.1276. As he is buying the currency, the consequence is that he has to pay a smaller amount of USD for the same amount of EUR. To be exact, he has to borrow and pay only $112 760, which makes a difference of $100 (112 860 − 112 760). This is called *positive slippage* because the trader's order gets filled at a better price than he required and thus the broker saves/makes money.

Scenario B: The market is highly volatile again, slippage occurs, however, this time in the opposite direction. The trader's buy order gets filled at 1.1296. As he is buying the currency, the consequence is that he has to pay a larger amount of USD for the same amount of EUR. He has to borrow and pay \$112 960 instead of the expected \$112 860 , which makes a difference of –\$100. This is called *negative slippage* because the trader had to enter the market at a worse price than he had expected. The consequence is a loss of profit that the trader would have obtained had his order been filled at the right time at the expected exchange rate.

It is important to note that slippage is a usual occurrence at the foreign exchange market. However, it is usually so small that it goes unnoticed. An exception is when the market is very volatile (for example due to news events). In that case, considerable slippage may occur, possibly incurring significant losses or profits.

Slippage is not impossible to fight. Examples of measures to limit the effect of slippage is the use of Limit orders or Market Range orders:

- *Limit order* – basically a Market order (a usual buy or sell order) with one difference. A Market order is always filled. However, because of slippage it might be filled at a worse price than requested. On the other hand, a Limit order is filled only in case the price is at least as good as the requested price. Otherwise, the order is not filled and stays in the waiting mode.

- *Market Range order* – similar to a Limit order. The difference is that, using this limit, the trader can set a range/amount by which the actual order price can be worse than the requested one. The greater the range, the higher the chance that the order will get filled. If the order is not filled, it is cancelled as a result, unlike the Limit order that goes back to the waiting mode.

Slippage should work both ways. Sometimes it results in a loss, sometimes in a profit. However, with some brokers the trader experiences only negative slippage. This is very unlikely to happen naturally. This might imply a dishonest behavior of the broker and it is better to avoid this kind of fraudulent brokers. To find out whether your potential broker belongs to this group or not, it is recommended to check the reviews and forums. The trader should be aware that these problems will hardly arise on demo accounts, as demo accounts tend to work with infinite liquidity.

1.2.14 Tax Reporting Support

This service is often overlooked by beginner traders. As the foreign exchange market is worldwide, unregulated and decentralized market brokers do not report trader's activity to tax authorities in the trader's country of residence.

It is a responsibility of the trader himself and if ignored, the trader might face tax evasion charges. Although brokers do not provide tax reports, all of them provide a detailed transaction history based on which the trader is able to create his tax report. However, the way how the transaction history is provided differs among brokers and might require additional effort when creating the tax report. Therefore, in the beginning of cooperation with a broker it is recommended to check if the transaction history provided by the broker meets the trader's requirements.

1.2.15 Trading Platform Options

Generally, trading platforms can be divided into the following three groups:

- Web
- Desktop software
- Mobile application

Some platforms cover more than one of these groups. For instance, MetaTrader 4 is considered to be one of the most famous platforms. It is used by a considerable number of brokers. It is available as both a desktop and a mobile application. Its successor MetaTrader 5 has not become so widespread yet. In some cases, brokers create their own proprietary platforms, such as Trading Station by the FXCM broker, which is a solution designed for web, desktop and mobile alike.

The main properties the trader should check when choosing his trading platform are the following:

- Expert Advisors (EA) support – in case the trader intends to automatize his trading using *Expert Advisors* (trading robots, algorithms), it is necessary to verify that the designated platform supports them.
- Charting – platforms also differ in charting capabilities, thus they might suit the trader's needs differently.
- Chart Trading – some platforms offer chart trading which, apart from the ability to open a trade directly from the chart, also imply a better overview of different orders, such as the stop-loss or the trailing stop, and the ability to see whether the market approaches triggering of these orders or not. Chart Trading is usually fast and comfortable.
- News Feed access – forex trading platforms often provide access to high-quality news feeds from professional forex market information sources that might include Reuters, Associated Press, Bloomberg or Telerate [13]. This is particularly useful when the trader builds his strategy on the fundamental analysis.

1.2.16 Virtual Private Server Service

A virtual private server (VPS) is a remote server where the trader can run his trading application. It has several advantages, such as:

- Reliability – most companies guarantee at least a 99.9% uptime. The usage of a VPS reduces the risk of a power outage and a hardware failure.
- Connection speed – usually VPS providers offer a higher and more stable connection speed than an average trader can achieve from his home. While the VPS service is offered by many IT companies, it is sometimes also offered by the brokers themselves (and hence the service is tailored for the requirements of algorithmic trading). In case the brokers do not outsource the VPS service and keep their VPS servers in a close distance to the main servers used for executing trades, the latency will be significantly lower as well. All of this can considerably reduce the amount of slippage, which might be expensive at times.

The VPS might be a very useful service in case the trader utilizes Expert Advisors for automatic trading. However, it is advantageous for traders who prefer manual trading as well, as the connection speed point applies to them as well. Additionally, the trader can connect to a VPS from anywhere regardless of the operating system being used. This adds another level of flexibility even to manual traders.

Not all brokers offer the VPS service and mostly not for all platforms. While some brokers offer the VPS service for free, others might charge money. For example, the FXCM broker offers the VPS service for trading using the MateTrader 4 platform for free as long as the trader has at least 5000 currency units in his account. In case his balance is lower, this particular broker charges 30 currency units per month for this service. FXCM does not offer the VPS service for demo accounts.

1.2.17 Trustworthiness and Customer Service

There is a huge amount of brokers available. Some of them are more reliable and trustworthy than others. It is not recommended to trust everybody with your money who claims to be trustworthy. Unsurprisingly, all of the brokers claim that they are trustworthy. The way how to, at least partially, confirm this claim is to check whether the broker is registered with the regulatory bodies. For performing this action it is necessary to know in which country the broker is located, as every country has its own regulatory bodies.

The following list contains several countries along with their corresponding regulatory bodies:

- **United States**: National Futures Association (NFA) and Commodity Futures Trading Commission (CFTC) [18] [8]

- **United Kingdom**: Financial Conduct Authority (FCA) and Prudential Regulation Authority (PRA) [12] [19]
- **Australia**: Australian Securities and Investment Commission (ASIC) [3]
- **Switzerland**: Swiss Federal Banking Commission (SFBC) [22]
- **Germany**: Bundesanstalt für Finanzdienstleistungsaufsicht (BaFIN) [7]
- **France**: Autorité des marchés financiers (AMF) [4]
- **Czech Republic**: Česká Národní Banka (ČNB) [10]

Although the fact that a broker is registered with the corresponding regulatory body ensures some level of security, it does not guarantee that there will be no problems with the broker. However, if a problem arises and the broker refuses to solve it, the trader always has the option to turn to these regulatory bodies and ask for help. In case a problem arises when dealing with a broker that is not registered with a regulatory body and the broker refuses to solve it, the trader has no option but to file a lawsuit against him. Therefore, it is not advisable to trade with unregulated brokers.

Another thing to consider when determining the trustworthiness of a broker is the customer service quality. Some brokers have an outstanding customer service when the trader is opening his account or is trading using a demo account. However, once the broker persuades the trader and obtains his money (after opening a live account), the quality of the customer service decreases rapidly. Note that this is not always the case. It is nevertheless recommended to conduct some research on the selected broker before opening a live account. Numerous reviews of brokers on different forums regarding the forex may become a source of such research. However, these forums are sometimes indirectly owned or sponsored by a broker. Therefore, it is necessary to search several forums of this type. The more, the better.

1.2.18 Other Parameters

FIFO

FIFO stands for "First In, First Out". It is a measure, introduced by the National Futures Association (NFA) in 2009, that forces the trader to close the oldest positions of a currency pair and unit size first before being able to close a younger position of the same currency pair and unit size. This measure has several consequences. Firstly, it makes hedging impossible. Additionally, it renders several trading strategies useless. It is obvious that this measure significantly restricts the trader's options to make a profit. However, as the NFA is the regulatory body of the USA, this measure is obligatory only to the brokers located in the USA. With the non-US brokers it is recommended to

make sure that they do not enforce FIFO. Generally, it is better to avoid this kind of brokers.

An example – the trader trades the EUR/USD currency pair. He decides to buy 100 000 €. In order to secure its investment he decides to hedge his position, i.e. open the opposing position, in other words, buying $130 000 (given the EUR/USD exchange rate is 1.3000). However, purchasing $130 000 means selling 100 000 €. Because of FIFO, the trader is not allowed to do this, as first he would have to sell 100 000 €he bought when he opened the first position. In other words, the trader would have to close his first position as the first step. Therefore, hedging is impossible.

Hedging

Hedging is one of the strategies to limit the losses when the trader enters into a trade with the intent of protecting an existing or anticipated position from an unwanted move in the foreign currency exchange rates [15]. The simplest hedge is the offsetting of one position with the opposite position within the same currency pair. The outcome is that in case the exchange rate increases, the trader gains on one position and loses on the other one. Similarly, when the rate goes down the trader loses on one position and profits on the other one. Therefore, if both positions are of the same size the trader does not incur any losses (with the exception of commission and spread) nor profits.

Not all brokers allow hedging. Therefore, if hedging is a part of the trader's strategy, he should verify that his chosen broker supports it.

Scalping

Scalping is one of the most popular foreign exchange strategies. It is based on executing many (up to a few hundred) fast trades per day. These trades are usually open only several seconds (a few minutes at most). Basically, they are closed as soon as the trade becomes profitable. Thus, the trader profits from tiny fluctuations in price. These profits are usually small, however, as trader executes tens to hundreds of such trades per day, the small profits add up and by the end of the day they might amount to a significant profit.

However, some brokers forbid scalping. One possible reason for this might be a slow execution time of a broker. By prohibiting scalping, the broker defends himself against arbitrage. Another reason might be a misleading behavior of brokers. Dealing Desk brokers (Market Makers) profit from the trader's losses and lose on the trader's profits. The minimization of the trader's profits is the incentive behind the scalping ban introduced by some deceptive brokers. It is generally advisable to avoid this kind of brokers.

PAMM Account Availability

PAMM stands for Percentage Allocation Management Module or Percentage Allocation Money Management. With a PAMM account, a person using it should be called an investor instead of a trader, as he does not trade himself, but he lets a chosen money manager offered by the forex broker to trade on his behalf. The reason for the name is that a single money manager usually has multiple different investors. He puts their money together into one pool. Each investor owns a percentage of the pool, depending on the amount of the invested funds. Eventually, the total profit or loss is split (after subtracting the money manager's charge in case of a profit) and credited or debited based on the percentage.

The broker displays the information regarding the performance of each of the money managers, their ratings and review so that the investor can make an informed choice.

This type of account is suitable for people who do not have time to study the foreign exchange trading, nevertheless, they would like to invest into it and eventually profit from it.

1.3 Broker Fees Bypass

The best way to lower broker's fees is by choosing a serious broker. A positive sign is when the broker specializes mainly on institutional traders instead of retail traders. These brokers are much more trustworthy. However, they require sizeable initial deposits and offer only standard accounts. Usually, with a larger trade size they charge lower commission (lower percentage).

The only real way to bypass broker's fees, and probably the most reliable way how to profit from the foreign exchange market trading, is to become a broker. The broker registration is administered by a regulatory body of the country where the broker resides (see chapter 1.2.17 for more details). For instance, if a company chooses to become a broker in Czech Republic, it must register at the Česká Národní Banka (Czech National Bank).

In the USA a prospective broker is required to take the following steps: [1]

1. Registration of the company in a local or offshore jurisdiction.
2. Application for a Forex broker license at the U.S. Security and Exchange Commission (SEC).
3. Opening a bank account within the jurisdiction to collect funds from clients.
4. An application for receiving online payments if the broker is to accept online funding.

5. Preparation of legal documents, including dealing manuals and agreements, anti-money-laundering policies, conflict of interest policies etc.

6. Pay registration fees and meet the minimum capital requirements for opening a brokerage business (Broker-dealer applicants and registrants must have and maintain the minimum net capital required by the Securities and Exchange Commission Rule 15c3-1 and comply with the SEC Rule 15c3-3 governing customer protection, reserves and custody of securities).

7. Find liquidity partners, clearing company(s) that will clear the broker's trades. (A *clearing company* will look for the broker to deliver a certain trading volume per month: e.g. a total amount of lots the broker's clients will be able to trade every month).

Every step will include a set of documents to be prepared as well as qualifications to be passed. The package of requirements will depend on the jurisdiction where the broker chooses to register his company.

Registrants must be prepared to pass a background investigation, pass examinations on general securities principles and state securities laws (NASD series 7, NASD series 63, 66). Among others, the following criteria are considered during registration application:

- Financial solvency
- Conviction of a crime
- Evidence of past inequitable or fraudulent business practices in the sale of securities

Chapter 2

Analysis

The analysis, and later the implementation chapter, will be dedicated to the process of a robot creation, starting with the basics of the MQL4 programming language, finishing with the implementation of a new robot. A suitable trading platform and broker will be designated in the following two subchapters.

2.1 Choice of Trading Platform

For a demonstration of how Expert Advisors (EAs) are programmed and used, I chose MetaTrader 4 from MetaQuotes Software Corp. – currently the most widespread platform for algorithmic trading among retail traders.

MetaQuotes Software Corp. has already released a successor of MetaTrader 4, named MetaTrader 5. There are a few advantages of MetaTrader 5, such as improved charting options, a polished graphical user interface, additional market indicators, built-in forex calendar, etc. However, MetaTrader 5 has several disadvantages. It implements FIFO (see chapter 1.2.18) and thus forbids hedging. Another inconvenience is the incompatibility with the MetaTrader 4 Expert Advisors and Market Indicators. However, huge a community around MetaTrader 4 created numerous Expert Advisors and Market Indicators over the years that are still commonly used. The fact that they are incompatible with MetaTrader 5 slows down the migration process. While converting the Expert Advisor from MQL4 to MQL5 is not a difficult task for an average programmer, many traders are mere EA users/consumers and are not able to reprogram them themselves. Because of the MetaTrader 4's popularity among traders, a great majority of brokers has stayed with this platform. Therefore, choosing MetaTrader 4 gives the trader many more options when choosing a broker.

2.2 Choice of Broker

In order to be able to test an EA on live data, it is necessary to select a broker. For this purpose, I tried to search for a broker with the following parameters in mind:

- MetaTrader 4 support
- ECN broker type
- Hedging allowed
- No FIFO
- Scalping allowed
- Preferably mini account
- Low stopout level
- Leverage of 1:100
- Trustworthy
- As many order types available as possible

On the other hand, I ignored the following parameters:

- News trading – it is a part of the fundamental analysis, but EAs use the mathematical analysis.
- PAMM – the goal is to show the workings of automatic trading, so having the option to invest money to a person who trades instead of us is not required.
- VPS – VPS service is normally not offered for demo accounts. Therefore, for our purposes it does not matter whether the broker offers it or not.
- Tax reporting – money that will be traded are virtual, which means no taxes.
- Slippage – does not occur on demo accounts.
- Available currency pairs – the Major currency pairs will be sufficient for our purposes.
- Withdrawal and deposit – no real currency is going to be deposited nor withdrawn.
- Minimum deposit – no real currency is going to be deposited.

- Swapfree account – not really necessary. Swap fees are a usual part of trading. Therefore, in order to show the effect of the swap fees on the final balance, I decided to skip the swapfree account option.

Based on the required parameters I decided for the British Atom8 foreign exchange broker. It supports MetaTrader 4, is of the ECN type, allows hedging and scalping, does not enforce FIFO, offers a mini account, has a stopout level of 50%, offers a leverage of 1:100 and is regulated by the British Financial Conduct Authority. I was not able to find the list of available order types. However, the basic ones will be sufficient in the beginning.

2.3 Simple Programming Guide

As was mentioned before, the best option to join the foreign exchange trade is by creating your own trading algorithm. This requires basic knowledge of programming as well as knowledge of different market indicators and the market itself.

Basic structures used for the robot programming will be introduced in this section. Additionally, the use of market signals will be elaborated.

Expert Advisors for MetaTrader 4 are programmed in the MQL4 programming language, which is based on the C language. However, with the release of build 600 on the 3rd February 2014, MQL4 received an update and it now supports many features of the C++ language, such as structures, classes, object pointers, *void* type, `this` keyword, encapsulation and extensibility of types, inheritance, polymorphism and typecasting. For more information regarding changes in build 600, refer to [16, Ch. What's New in MQL4]. In the following sections only features specific for MQL4 will be discussed. Therefore, some previous knowledge of C++ is required.

One of the C++ language features that are still not supported in build 600 are class templates. However, function templates work well. Also, the pointers work in a slightly different way.

2.3.1 The Big Picture

Figure 2.1 depicts the entire trading system structure and the place of EAs in it. [21, Ch. Introduction to MQL4 programming]

The *dealing center* is basically the broker server chosen by the trader. It communicates with the rest of the market participants, sends the current market state information to the client's *terminal* and receives orders from it (the green line in the picture). The communication with the market participants is dependent on the Internet connection of the trader. A stable Internet connection is crucial for a timely order execution and for obtaining the most current market information.

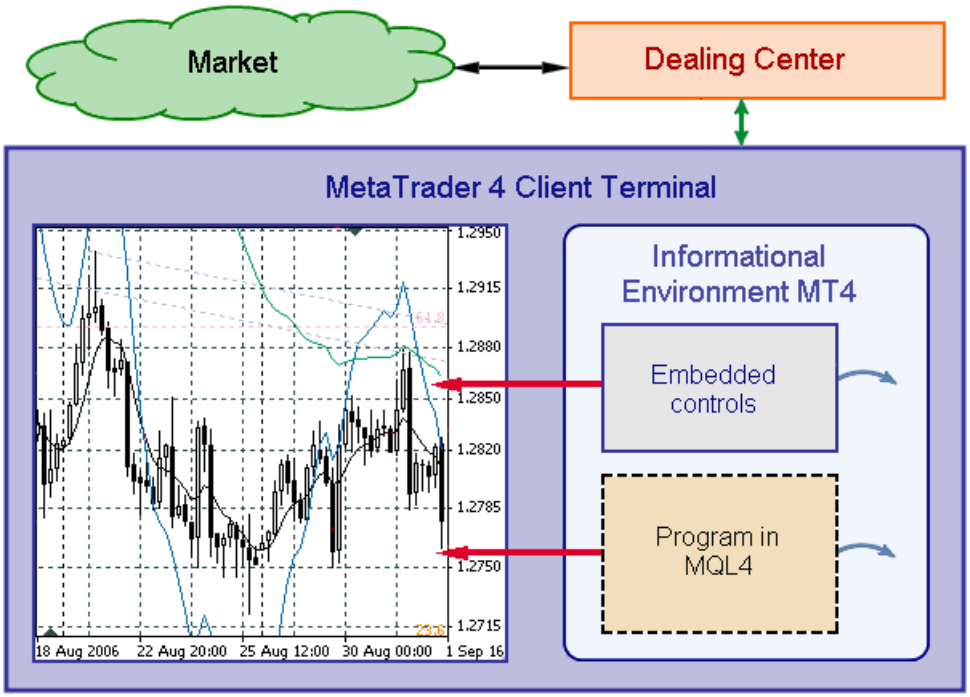

Figure 2.1: Trading system structure

The *client terminal* includes an *informational environment*, a set of parameters with information about the market state and about the relations between the trader and the dealing center. These parameters include the information about the current prices, the limitations on the maximum and minimum order size, the minimum distance of stop orders, the allowance and prohibition of the automated trading, and many other useful parameters characterizing the current state. The *informational environment* is updated when new ticks are received by the *terminal* (corresponding to the green line in Figure 2.1). [21, Ch. Introduction to MQL4 programming]

The *embedded controls* allow the trader to conduct a technical analysis of the market and to execute manual trading management. The trader's actions with the built-in trading management tools result in a formation of trade orders, which are sent through the *terminal* to the *dealing center*. [21, Ch. Introduction to MQL4 programming]

The market analysis and trade management in MetaTrader 4 *client terminal* is implemented with the help of *programs* [21, Ch. Introduction to MQL4 programming]. Three types of programs can be created:

1. Custom Market Indicators – display controls aiding the market analysis

2. Expert Advisors – perform automated trading
3. Scripts – execute one-time actions

This guide will focus mainly on the implementation of Expert Advisors.

Figure 2.1 shows that the *program* has the same means of access to the *client terminal informational environment* as the *embedded controls* for manual trading (displayed as blue arrows). It can also form managing influences (portayed as red arrows), passed to the *client terminal*. *Programs* of different types can be used simultaneously and they can exchange data. [21, Ch. Introduction to MQL4 programming]

2.3.2 Data Types

The basic data types supported by MQL4 originally were *int*, *double*, *bool*, *string*, *color* and *datetime* [21, Ch. Data types]. Since MetaTrader 4 build 600 the language provides support for the *char*, *short*, *long*, *uchar*, *ushort*, *uint*, *ulong* and *double* data types [16, Ch. What's New in MQL4]. Additionally, as the creation of classes was added as well, it is posibble to define a custom data type.

With the exception of *color* and *datetime*, all the above types correspond to the C++ types of the same name. Consequently, only *color* and *datetime* will be discussed:

1. *color* – the values of color constants and variables can be represented as one of the three kinds: literals, integer representations and color names [21, Ch. Data types]:

   - Literal – the value of the color type represented as a literal consists of three parts representing the numeric values of the intensity of the three basic colors: red, green and blue (RGB). The value of this kind starts with a 'C' and is quoted by single quotes.

     The numeric values of the RGB intensity range from 0 to 255, and they can be recorded both decimally and hexadecimally.

     Examples: C'128,128,128' (gray), C'0x00,0x00,0xFF' (blue). [21, Ch. Data types]
   - Integer representation – recorded as a hexadecimal or a decimal number. A hexadecimal number is displayed as 0xRRGGBB, where RR is the value of red intensity, GG of green and BB of blue. Decimal constants are not directly reflected in RGB. They represent the decimal value of a hexadecimal integer representation.

     Examples: 0xFFFFFF (white), 0x008000 (green). [21, Ch. Data types]

- Color name – last but not least, one can use predefined color names. These include *Yellow*, *Green*, *MediumTurquoise*, *Magenta*, etc. For the full list of predefined colors see section named *Color names* at [21, Ch. Data types].

2. *datetime* – as the name suggests, this type is used to store both date and time. Time and date constants "start with letter 'D' and are framed in single quotes. It is also possible to use truncated values without date, or without time, or just as an empty value. The range of values is from January 1, 1970 through December 31, 2037. The values of constants and variables of datetime type take 4 bytes in the computer memory. A value represents the amount of seconds elapsed from 00:00 of January 1, 1970" [21, Ch. Data types].

As for arrays, "all arrays are static, i.e. are of static type even if at the initialization this is not explicitly indicated. It means all arrays preserve their values between calls of the function, in which the array is declared" [21, Ch. Arrays]. The maximum array dimension number allowed by MQL4 is 4.

2.3.3 File Types and Locations

The following file types are specific to MetaTrader 4:

- *mq4* – source code (of an EA, a script or a custom indicator)
- *ex4* – compiled executable from *mq4*, can also be used as a library
- *mqh* – header (include) files (included with the `#include` directive, usually residing in *DataFolder/MQL4/Include*) used for including frequently used blocks of source code

Files for EAs reside in the *DataFolder/MQL4/Experts* folder; custom market indicators are located in the *DataFolder/MQL4/Indicators* folder, while scripts can be found in *DataFolder/MQL4/Scripts*.

It is also possible to create or utilize libraries for reusing of pieces of code (these are located in the *DataFolder/MQL4/Libraries* folder). However, it is advisable to use include files instead, as they are linked into the final executable at compile time, while libraries are linked and called at run-time.

The easiest way to locate MetaTrader's *DataFolder* is by running the terminal and selecting *Open Data Folder* from the *File* menu.

2.3.4 Creating a New Expert Advisor

For programming new EAs (or for that matter, scripts, custom indicators, include files and libraries), the MetaEditor is used. It is located in the MetaTrader's installation folder, next to the executable that runs the trading terminal. If you have the MetaTrader terminal running, the easiest way to invoke

the MetaEditor is by selecting *Tools/MetaQuotes Language Editor* in the menu or by pressing F4. Another option is to right-click the *Expert Advisors* in the Navigator control and choose *Create in MetaEditor*.

In MetaEditor, clicking the *New* button in the toolbar runs a wizard for creating a new file. From the available types choose *Expert Advisor* and proceed with the next steps of the wizard until finished. If you do not know whether you need to add certain events or inputs during the wizard or not, you can simply skip them. They can be added later directly into the code if required.

Listing 2.1

```
1  //+------------------------------------------------------------------+
2  //|                                                   skeleton.mq4 |
3  //|                                                   Juraj Korcek |
4  //|                                          https://www.mql5.com |
5  //+------------------------------------------------------------------+
6  #property copyright "Juraj Korcek"
7  #property link      "https://www.mql5.com"
8  #property version   "1.00"
9  #property strict
10 //--- input parameters
11 input int       sample_input;
12 //+------------------------------------------------------------------+
13 //| Expert initialization function                                   |
14 //+------------------------------------------------------------------+
15 int OnInit()
16 {
17     //---
18
19     //---
20     return(INIT_SUCCEEDED);
21 }
22 //+------------------------------------------------------------------+
23 //| Expert deinitialization function                                 |
24 //+------------------------------------------------------------------+
25 void OnDeinit(const int reason)
26 {
27     //---
28
29 }
30 //+------------------------------------------------------------------+
31 //| Expert tick function                                             |
32 //+------------------------------------------------------------------+
33 void OnTick()
34 {
35     //---
36
37 }
38 //+------------------------------------------------------------------+
```

Listing 2.1 represents a basic structure (without additional events) of an EA. A single input was added in the wizard in order to clarify where

the program inputs should be defined.

This basic structure can already be compiled. In order to do so, click the *Compile* button in the MetaEditor toolbar or press F7. During the compilation, the source code of the EA (the *mq4* file) is taken and checked that it is syntactically and semantically correct, and compiled into an *ex4* file afterwards. Unless an error occurs, the EA will appear in the MetaTrader terminal's Navigator control in the *Expert Advisors* category.

In order to use this EA, it is necessary to drag-and-drop it to the required currency pair window, double-click it, or right-click it and choose the *Attach to a chart* option. The name of the currently running EA is shown in the upper right-hand corner of the currency pair window. It has either a smiling or a frownin face attached to its name. The face depicts whether the EA was enabled for live trading or not. A frowning face means live trading is disabled for this EA and therefore it can be used with historical data in the strategy tester. A smiling face means live trading is allowed for this particular EA. To enable or disable live trading either double-click the face next to the name of the EA in the upper right-hand corner of the currency pair window or right-click the chart and choose *Expert Advisors/Properties...* or press F7. When the properties dialog opens, navigate to the *Common* tab and enable/disable the *Allow live trading* checkbox.

All the errors and messages from the EA will be shown in the *Journal* tab of the terminal control.

Even though the EA is now running (given it is attached to the chart, live trading is allowed and the market is open), nothing is happening. This is because we are running only the base EA structure which contains no instructions for trading so far.

In order to remove the EA from the currency pair window, right-click any part of it and choose *Expert Advisors/Remove* from the menu.

2.3.5 Execution Order

The program code consists of several distinct parts. To be specific, the parts are the *head*, the *OnInit()* special function (*init()* before the changes in build 600), the *OnTick()* special function (*start()* before build 600), the *OnDeinit()* special function (*deinit()* before build 600) and any user-defined functions. Additionally, both the special functions and the user-defined functions can use the standard functions of MQL4.

1. Head – this part is always the first one. It consists of the definitions of properties, the definition and initialization of input, the extern and global variables as well as the inclusion of libraries and include files. The variables generally need to be defined before they are used in the program. To ensure this property, the variables are declared in the beginning of the program.

2. *OnInit()* – a special function called at the initialization of a given EA. That is, "after a financial instrument and/or chart timeframe is changed, after a program is recompiled in MetaEditor, after input parameters are changed from the setup window of an Expert Advisor or a custom indicator. An Expert Advisor is also initialized after the account is changed" [16, Ch. Client Terminal Events]. It is typically used for the initialization of global variables.

3. *OnTick()* – this special function "is generated if there are new quotes. In case when the *OnTick()* function for the previous quote is being processed when a new quote is received, the new quote will be ignored by an Expert Advisor, because the corresponding event will not enqueued" [16, Ch. Client Terminal Events]. All the trade execution takes place here.

4. *OnDeinit()* – another special function. It is triggered when the current EA is unloaded. It is also invoked "when the client terminal is closed, when a chart is closed, right before the security and/or timeframe is changed, at a successful program re-compilation, when input parameters are changed, and when account is changed. The deinitialization reason can be obtained from the parameter, passed to the *OnDeinit()* function. The *OnDeinit()* function run is restricted to 2.5 seconds. If during this time the function hasn't been completed, then it is forcibly terminated" [16, Ch. Client Terminal Events]. It can be used for cleaning allocated resources.

5. User-defined functions – a special category of functions that are defined by the user himself. They can be called from the special functions, such as *OnInit()*, *OnTick()* or *OnDeinit()*. They are helpful for making the code more readable by separating functionality into separate blocks.

Another category of functions are the standard functions. These are built-in functions, therefore, they cannot be redefined. They can only be called from the program parts 2 to 5 (see the previous list). The overall program structure and the invocation of respective parts is portrayed in Figure 2.2. [21, Ch. Program structure]

2.3.6 Predefined Variables

The values of all predefined variables are automatically updated by a client terminal at the moment when the special functions are started for execution. [21, Ch. Predefined Variables and RefreshRates Function]

A list of simple predefined names of variables [21, Ch. Predefined Variables and RefreshRates Function]:

- Ask – last known sell-price of a current security

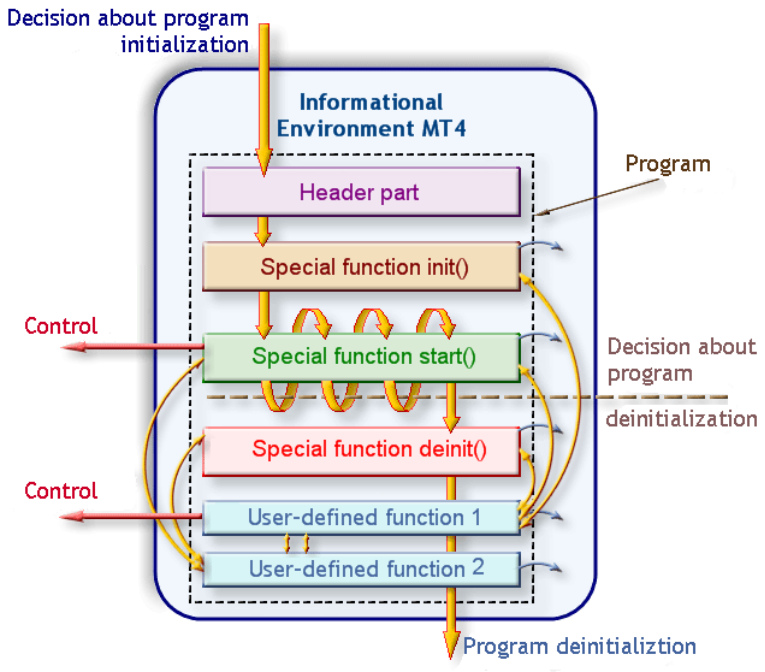

Figure 2.2: Program structure

- Bid – last known buy-price of a current security
- Bars – number of bars on a current chart
- Point – point size of a current security in quote currency
- Digits – number of digits after a decimal point in the price of a current security

A list of predefined names of arrray timeseries [21, Ch. Predefined Variables and RefreshRates Function]:

- Time – opening time of each bar on the current chart
- Open – opening price of each bar on the current chart
- Close – closing price of each bar on the current chart
- High – maximal price of each bar on the current chart
- Low – minimal price of each bar on the current chart
- Volume – tick volume of each bar on the current chart

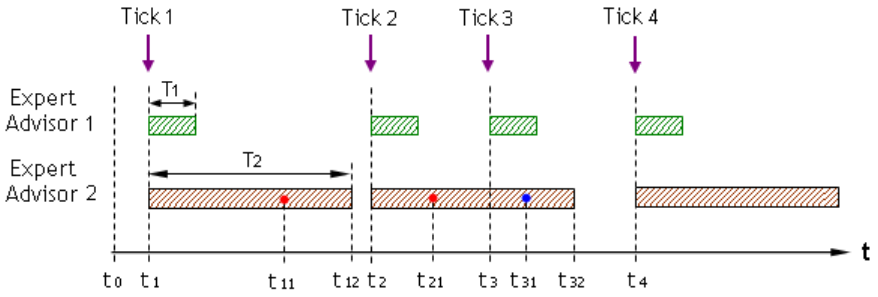

Figure 2.3: Array timeseries

For array timeseries the item at the zeroth index represents the newest bar, the one at the first index represents the second newest bar, etc. This is depicted in Figure 2.3. [21, Ch. Predefined Variables and RefreshRates Function]

If a new tick comes during the execution of the current one, it will be ignored. However, this means that the values of the predefined variables, which the *OnTick()* function is working with, are outdated, which can result in incorrectly defined trades which will eventually be rejected. To avoid this, the *RefreshRates()* function can be used inside the *OnTick()* function to update these variables to most up-to-date values.

These predefined variables store only the basic information regarding the current state of the market. However, using *MarketInfo()* it is possible to obtain all the information in the predefined variables and additional pieces of information, such as swap type, minimum lot and others. *MarketInfo()* always returns the current values without the need to call *RefreshRates()* first.

2.3.7 External and Input Variables

External and *input* variables are supposed to be defined in the head of the program. They define the input variables of the program. The external variables are set with the `extern` keyword, while the input variables are set with the `input` keyword. The only difference between these two is that the input variables are read-only, i.e. they cannot be modified inside the EA, while it is allowed to modify the external variables.

Both input variables can be changed from the user interface, particularly in the program properties toolbar, see Figure 2.4. However, "it must be remembered that program properties toolbar can be opened only in the period when the program (Expert Advisor or Indicator) is waiting for a new tick, i.e. none of special functions is executed. During the program execution period this [toolbar] cannot be opened." [21, Ch. Types of Variables]

Also, it is important to remember that changing these variables from the user interface restarts the EA. More specifically, "the client terminal starts successively the execution of the special function *deinit()*, then the special function *init()*, after that when a new tick comes – *start()*. At the execution

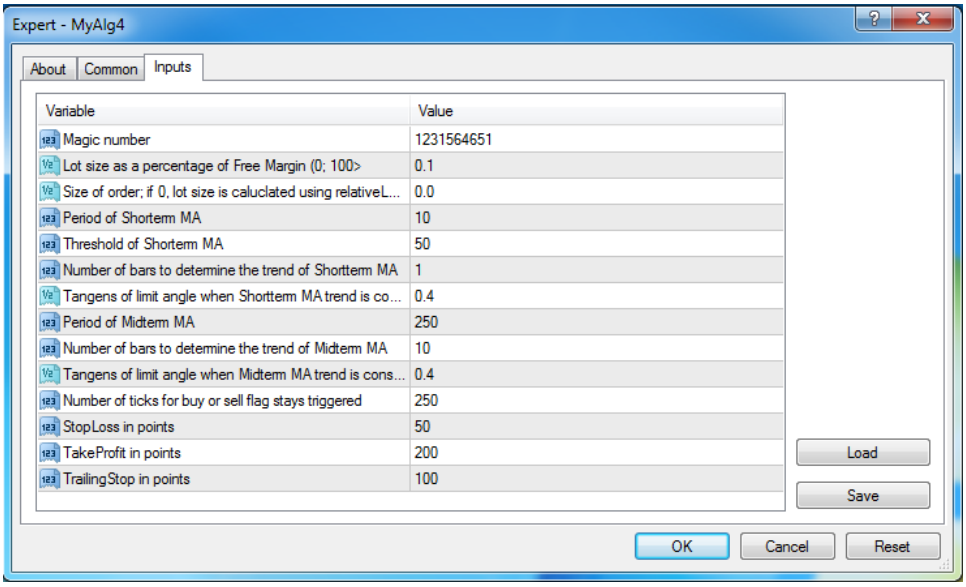

Figure 2.4: Input parameter settings

of *deinit()* that finishes a program, external variables will have values resulted from the previous session, i.e. those available before the EA settings [toolbar] was opened. Before the execution of *init()* external variables will get values setup by a user in the settings toolbar and at the execution of *init()* external variables will have new values set by a user. Thus new values of external variables become valid from the moment of a new session (*init* – *start* – *deinit*) of an Expert Advisor that starts from the execution of *init()*." [21, Ch. Types of Variables]

2.3.8 Global Variables with Respect to the Environment

This section discusses the use of the *global* variables. However, not the variables global to the program (i.e. those defined in the head of the program), but the variables global to the whole trading environment. That means they can be accessed by any EA, custom Market Indicator or script running in the terminal. To avoid confusion, these global variables will be referred to as *environment variables* throughout the rest of the thesis.

As distinct from other variables, environment variables can be not only created from any program, but also deleted. Environment variable value is stored on a hard disk and saved after a client terminal is closed. Once declared, environment variable exists in the client terminal for 4 weeks from the moment of the last call. If during this period none of programs called this variable, it is deleted by the client terminal. Environment variables can be only of double type. [21, Ch. GlobalVariables]

Basic methods for working with the environment variables are the following:

- `datetime GlobalVariableSet(string name, double value)`

  If a variable with the given name exists, it updates its value, otherwise it creates a new environment variable with the given name and initializes it to the desired value.

- `double GlobalVariableGet(string name)`

  Retrieves the value of an environment variable.

- `bool GlobalVariableDel(string name)`

  Tries to delete a environment variable and returns *true* in case of success, *false* otherwise.

For more available methods for working with environment variables see [16, Ch. Global Variables of the Client Terminal].

2.3.9 Trade Operations and Order Management

The basic functions used for managing trades and positions are the following:

- `int OrderSend(string symbol, int cmd, double volume, double price, int slippage, double stoploss, double takeprofit, string comment=NULL, int magic=0, datetime expiration=0, color arrow_color=clrNONE)`

  This function is used to open market and pending orders. *Volume* is set in lots. *Slippage*, *stopLoss* and *takeprofit* are set in points. The *magic* parameter is for setting the magic number. The *expiration* parameter is used only for the pending orders. If the order is not executed before this time, the order will be cancelled. The *arrow_color* parameter sets the color of the arrow on the chart. *Cmd* can be one of those in the table 2.1.

  It returns the number of the ticket as assigned by the server or -1 in case of an order open failure. [16, Ch. OrderSend]

- `bool OrderClose(int ticket, double lots, double price, int slippage, color arrow_color)`

  A function used to close an order. The *ticket* parameter is a unigue number by which the order to be closed is determined. Number of *lots* does not have to be necessarily equal to the amount of lots of the open order. If the amount is lower, the order will be closed only partially. The rest of the parameters bahave in the same way as their equivalents in the *OrderSend()* function. *OrderClose()* returns *true* in case of success,

Table 2.1: Possible values of the *cmd* parameter of the *OrderSend()* function

OP_BUY	0	Buy
OP_SELL	1	Sell
OP_BUYLIMIT	2	Pending order BUY LIMIT
OP_SELLLIMIT	3	Pending order SELL LIMIT
OP_BUYSTOP	4	Pending order BUY STOP
OP_SELLSTOP	5	Pending order SELL STOP

false otherwise. An attempt to close a pending order will result in an error, as pending orders can be deleted but not closed.

- `bool OrderCloseBy(int ticket, int opposite, color arrow_color)`

  This function closes a market order by another one of the opposite type (i.e. a buy order can be closed by a sell order or vice versa). [16, Ch. OrderCloseBy] In this way, the pips amounting to one spread can be saved. The *ticket* parameter takes the number of one order, the *opposite* parameter takes the number of the second order. Returns *true* in case of success, *false* otherwise. An attempt to close a pending order will result in an error, as pending orders can be deleted but not closed.

- `bool OrderDelete(int ticket, color arrow_color)`

  Deletes a pending order (determined by the *ticket* number) which has not been executed yet. An attempt to delete a market order will result in an error, as only the deletion of pending orders is allowed. Returns *true* in case of success, *false* otherwise.

- `bool OrderModify(int ticket, double price, double stoploss, double takeprofit, datetime expiration, color arrow_color)`

  A function used to modify a previously open market or pending order. *Open price* and *expiration time* can only be changed for pending orders. Returns *true* in case of success, *false* otherwise. [16, Ch. OrderModify]

Only one request is possible in time per terminal. When trade request is created by EA, no other trade request is allowed by any EA regardless of the currency pair until the original request is either rejected by the terminal, rejected by the server or accpeted by the server. [21, Ch. Common Way of Making Trades]

When opening, modifying and deleting orders, it is necessary to pay attention to the minimum and freeze distance in order to avoid errors.

The *minimum distance* is a parameter set by the broker and it sets the minimum distance in points from current price for setting *StopLoss*, *TakeProfit* and for opening pending positions. If the user tries to set *StopLoss*, *TakeProfit*

or to open pending positions closer to the current price than this distance, the order will be rejected. To avoid this kind of errors it is recommended to obtain the minimum distance value using the function *MarketInfo(Symbol(), MODE_STOPLEVEL)* and adjust StopLoss, TakeProfit and the open price for opening pending positions accordingly. [21, Ch. Event Tracking Function]

Similarly, the *freeze distance* is set by the broker. Modifying an order is forbidden if the current price is closer to the order than this distance. Therefore, in that case the calls to functions *OrderModify()* and *OrderDelete()* will result in an error. To avoid these errors it is recommended to check the current value of the freeze distance and according to that to call or not to call the aforementioned functions. The current value of the freeze distance is obtained by calling *MarketInfo(Symbol(), MODE_FREEZELEVEL)*. [21, Ch. Requirements and Limitations in Making Trades]

In order to access the information about a given order it is required to load the order first by using the *OrderSelect()* function. The signature of *OrderSelect()* is the following: `bool OrderSelect(int index, int select, int pool=MODE_TRADES)`.

The *select* paramter sets the selection mode, which can be either selection by position (SELECT_BY_POS) or selection by ticket (SELECT_BY_TICKET). Based on this flag, the *index* parameter should be set either to index (with 0 being the newest trade in case of SELECT_BY_POS) or to ticket number (in case of SELECT_BY_TICKET). The *pool* parameter defines weather to search in the current open and pending orders (MODE_TRADES) or in the history – closed and cancelled orders (MODE_HISTORY). If *select* is set to SELECT_BY_TICKET, the *pool* parameter is ignored and the order is searched by tick in both categories. Returns *true* on success, *false* otherwise. Possible reasons for failure are, for example, a nonexistent ticket order or a postion higher than number of orders. [16, Ch. OrderSelect]

Once an order was selected by using the *OrderSelect()* function, the trader can obtain information about the order by subsequent calls to these functions: *OrderClosePrice()*, *OrderCloseTime()*, *OrderComment()*, *OrderCommission()*, *OrderExpiration()*, *OrderLots()*, *OrderMagicNumber()*, *OrderOpenPrice()*, *OrderOpenTime()*, *OrderPrint()*, *OrderProfit()*, *OrderStopLoss()*, *OrderSwap()*, *OrderSymbol()*, *OrderTakeProfit()*, *OrderTicket()*, *OrderType()*. [16, Ch. OrderSelect]

2.3.10 Use of Market Indicators

Market indicators are a useful tool for aiding trading decisions. Many contemporary strategies consist of combined use of various indicators. MetaTrader4 has two categories of market indicators – technical and custom. [21, Ch. Usage of Technical Indicators]

Technical Indicators

The *technical indicators* are indicators that are built-in by default. Their names usually start with an 'i'. For example, iMA (Moving average), iSAR (Parabolic Stop And Reverse System) or iStochastic (Stochastic Oscillator) are very popular. For list of all available technical indicators see appendix. [21, Ch. Usage of Technical Indicators]

Following is an example of the setup and use of iMA indicator.

This technical indicator calculates moving average. It has the following signature:

Listing 2.2

```
1  double iMA(
2        string symbol,        // symbol
3        int timeframe,        // timeframe
4        int ma_period,        // MA averaging period
5        int ma_shift,         // MA shift
6        int ma_method,        // averaging method
7        int applied_price,    // applied price
8        int shift             // shift
9  );
```

Therefore, if the trader wants to obtain the value of the moving average at the last bar for the current symbol and the current timeframe (the timeframe of the window the EA is attached to) with the period set to 10 bars, no shift, using the simple averaging method on the close price, the call will look like: `MA=iMA(NULL, 0, 10, 0,MODE_SMA,PRICE_CLOSE,0)`.

If NULL is passed as the *symbol* parameter, the function will automatically use the symbol of the window in which the EA is running. It is equivalent to passing *Symbol()* as the parameter. If 0 is passed to the *timeframe* parameter, the function will automatically use the timeframe of the window the EA is running in. It is equivalent to passing *Period()* as the parameter.

In this case the averaging method is the *simple* one (MODE_SMA). Additional averaging methods are *exponential* (MODE_EMA), *smoothed* (MODE_SMMA) and *linear-weighted* (MODE_LWMA). [16, Ch. Smoothing Methods]

The applied price can be one of seven possible values. The basic four values are *close* (PRICE_CLOSE), *open* (PRICE_OPEN), *high* (PRICE_HIGH) or *low* (PRICE_LOW) price of the bar. The other three values are a combination of the previous four. *Median* price (PRICE_MEDIAN) is $(high + low)/2$, *typical* price (PRICE_TYPICAL) is $(high + low + close)/3$ while *weighted* price (PRICE_WEIGHTED) is $(high + low + 2 \times close)/4$. [16, Ch. Price Constants]

Every indicator has a different set of parameters. To see indicators other than iMA refer to the official documentation.

Custom Indicators

The second group of the market indicators available in MetaTrader4, the *custom indicators*, is user-made. Custom indicators are called with iCustom function, where one of its parameters is the name of the custom user-defined indicator. [21, Ch. Usage of Technical Indicators]

The signature of the function is `double iCustom(string symbol, int timeframe, string name, ..., int mode, int shift)`, where *name* is the name of the custom indicators and ... stands for all the parameters it requires.

The custom indicator must be compiled (*.ex4* file) and located in the directory *Terminal_data_folder\MQL4\Indicators*. [21, Ch. Combined Use of Programs]

The execution of *iCustom()* does not require the attachment of a corresponding indicator to a security window, as well as the call of *iCustom()* from any application program does not result in the attachment of a corresponding indicator to a security window. The attachment of a technical indicator to a security window also does not lead to the call of *iCustom()* in any application program. [21, Ch. Combined Use of Programs] The same rules apply to the built-in technical parameters.

2.3.11 Permanent Run

One of the options is a program running permanently. That means that it does not get activated with every new tick, but it runs in a loop until the EA is closed. As new *OnTick()* events are not received in this case, it is necessary to use the *RefreshRates()* function to update the predefined variables with the most up-to-date values.

Example of such a function [21, Ch. General Characteristics of Complex Programs]:

Listing 2.3

```
1  void OnTick()    // Special function Tick()
2  {
3      // Until user stops execution of the program
4      while (!IsStopped())
5      {
6          RefreshRates();      // Data renewal
7          // ... The main code of the program is specified here
8          Sleep(5);            // Short pause
9      }
10     return;      // Control is returned to the terminal
11 }
```

2.3.12 External Library Usage

It is also possible to import and call functions from any *Dynamic-link library* (DLL). This has a few advantages. A DLL can be written in any language

and any editor. The programmer can therefore use tools that he is used to and avoid using the incomplete MQL4 language and not so user-friendly MetaEditor. All this can increase effectiveness of the EA programming significantly. Additionally, in this way, multithreading, database connection and other useful constructs or tools can be used to create advanced (for example machine learning) algorithms which would be impossible to program in MQL4/MetaEditor.

The idea is to initialize a library in the *OnInit()* function. Then, in the *OnTick()* function send all the current account and market data and received trading commands from it. In the *OnDeinit()* function the library will be deinitialized.

As a DLL can run asynchronously, the response to the function calls can be immediate. This means that the execution of the *OnTick()* function will be very fast, which considerably increases the chance that only the absolute minimum of ticks are missed.

An example of a DLL being called from the EA:

Listing 2.4

```
1  #import "my_expert.dll"
2  int ExpertRecalculate(int wParam, int lParam);
3
4  void OnTick()        // Special function Tick()
5  {
6      ExpertRecalculate(2, 1);
7      return;          // Control is returned to the terminal
8  }
```

2.3.13 Other Useful Functions

This section lists some of the most used standard functions with the explanations of their functionality.

- `double AccountFreeMargin()`

  Returns the free margin of the current account [16, Ch. AccountFreeMargin]. It is usually used for determining the upper limit for the order size.

- `double AccountFreeMarginCheck(string symbol, int cmd, double volume)`

  Returns the remaining free margin given that the order of the type *cmd* is executed with the volume specified by the *volume* parameter. If lower than 0, then the free margin is insufficient for executing this order. [16, Ch. AccountFreeMarginCheck]

- `int ArrayMaximum(const void& array[], int count=WHOLE_ARRAY, int start=0)`

This function returns the index of the biggest element in a numeric array. It can also be specified from which element to start searching and how many elements to search. If these parameters are not specified, the whole array is searched. [16, Ch. ArrayMaximum]

- `int ArrayMinimum(const void& array[], int count=WHOLE_ARRAY, int start=0)`

  Works in the same way as the *ArrayMaximum()* function with the only difference being the fact that it returns the index of the element with the lowest value in the array.

- `void Comment(argument, ...)`

  This function displays a message in the upper left corner of the chart window. It can be used to communicate the status of the EA to the user. It takes up to 64 arguments which are shown on separate lines. The limit for the whole comment is 2045 symbols. [16, Ch. Comment]

- `int GetLastError()`

  Returns the value of the last error that occurred during the execution of a MQL4 program. After the call of the function, the last error value is reset to 0. For the list of the trade server return codes see [16, Ch. Trade Server Return Codes]. [16, Ch. GetLastError]

- `double NormalizeDouble(double value, int digits)`

  Rounds a floating-point number (the *value* parameter) to the number of decimal digits given by the *digits* parameter. [16, Ch. NormalizeDouble]

- `int OrdersTotal()`

  Returns the number of the open market and pending orders. [16, Ch. OrdersTotal]

- `int Period()`

  Returns the current chart timeframe as an integer which can be checked against the periods from ENUM_TIMEFRAMES (e.g. PERIOD_M1=1, PERIOD_H1=60). For the list of all possible timeframes see [16, Ch. Chart Timeframes]. It is often used as a parameter for the market indicator functions. [16, Ch. Period]

- `void Sleep(int milliseconds)`

  The function stops the execution of the program for the given amount of milliseconds. It cannot be called from the custom market indicators. The function call is ignored during a run in the Strategy Tester. [16, Ch. Sleep]

- `string Symbol()`

  "This function returns a string value that corresponds with the name of the symbol, in the window of which the EA or script is being executed." [16, Ch. Opening and Placing Orders]

2.3.14 Control

Once the trader attached a program (an EA or a custom indicator) to the symbol window, the program makes some preparations and switches to the tick-waiting mode. "As soon as a new tick comes, the program is launched by the client terminal for execution. Then it makes all necessary operations prescribed by its algorithm. Upon completion, the program passes the control to the client terminal, i.e. it switches to the tick-waiting mode." [21, Ch. Program Types]

"If a new tick comes when the program is being executed, this does not have an effect on the program execution – the program continues being executed according to its algorithm, and the program passes the control to the client terminal only upon completion. This is why not all the ticks result in launching of an EA or a custom indicator. Only those ticks that come in when the control is in the client terminal and when the program is in the tick-waiting mode call the EA or the custom indicator." [21, Ch. Program Types]

"The new tick launches the program for execution. Thus, an EA or a custom indicator can operate within a long period of time, being attached to the symbol window and starting to run from time to time (as often as a new tick comes in while the program is in the tick-waiting mode)." [21, Ch. Program Types]

The difference between EAs and custom indicators is in the first-time execution. While in case of the EAs *OnInit()* is called immediately and *OnTick()* is then called after receiving the first tick, in case of the market indicators both *OnInit()* and *OnTick()* are launched immediately after attaching the indicator to the chart window. [21, Ch. Program Types]

2.4 Analysis of Publicly Available Robots

Out of the EAs publicly available on the official website ([17]) I have chosen *Geedo* and *Genie* for analysis. These robots have numerous input parameters and therefore many different setup configurations to test and choose from.

2.4.1 Geedo

This algorithm has only two input parameters (*TakeProfit_S* and *TradeTime*) by default. However, by inspecting the head of the program many more parameters were found. In order to obtain more options to optimize the algorithm

I made all of them available by adding the `extern` keyword before their declaration. The additional parameters are *TakeProfit_L*, *StopLoss_L*, *StopLoss_S*, *t1*, *t2*, *delta_L*, *delta_S*, *lot*, *Orders*, *MaxOpenTime*, *BigLotSize*.

Head

In the head the algorithm contains the initialization of the parameters and the declaration of the global variables that are used later in some of the functions.

Listing 2.5: Geedo input parameters

```
1  extern int TakeProfit_L = 39;     // Take Profit in points
2  extern int StopLoss_L = 147;      // Stop Loss in points
3  extern int TakeProfit_S = 15;     // Take Profit in points
4  extern int StopLoss_S = 6000;     // Stop Loss in points
5  extern int TradeTime=18;          // Time to enter the market
6  extern int t1=6;
7  extern int t2=2;
8  extern int delta_L=6;
9  extern int delta_S=21;
10 extern double lot = 0.01;         // Lot size
11 extern int Orders=1;              // Maximal number of posi-
12                                   // -tions opened at a time
13 extern int MaxOpenTime=504;
14 extern int BigLotSize = 1;        // By how much lot size is
15                                   // multiplicated in Big lot
```

Listing 2.6: Geedo variables

```
1  int ticket, total, cnt;
2  bool cantrade=true;
3  double closeprice;
4  double tmp;
```

globPos()

The first function, *globPos()*, increases the value of the environment variable *globalPosic* by 1. However, after inspecting the usage of this function I found out that it is useless, as always after calling this function the environment variable *globalPosic* is set to 0 by calling `GlobalVariableSet("globalPosic",0)`.

OpenLong()

As its name suggests, the function *OpenLong()* takes care of opening a long position. It takes the size of the position in lots as a parameter.

In the beginning some of the constants are set. Maximum allowed slippage is set to 10. The comment to distinguish the long position is set. It is usually good practice to set the comment to the name of the EA, so that the trader can easily distinguish which trades were opened by him manually and which were opened by the EA. Setting the arrow color is necessary to draw an arrow

into the chart. If no color is set, no arrow is drawn. It is also good practice to distinguish different market orders opening and closing with different colors. The magic number is set to 0. This is fine if we are sure that our EA is the only one running for the given currency pair and that no manual trading takes place. However, it is usually set to a high number specific to the EA. This way it can be used to make sure that EA works only with orders which it itself created.

The line 9 in Listing 2.7 checks the value of global variable *globalBalans* and if it is more then the actual balance, the volume is increased by the *BigLotSize* constant at line 10. However, given that this EA is the only one using the *globalBalans* environment variable, and as *globalBalans* is updated after opening of every order, the only case when this condition is *true* is when an order is closed with a loss. Therefore, the result of these two lines is that after a losing trade the volume traded is increased, which is slightly unanticipated. Testing of this algorithm will show the extent to which this condition is useful.

The statement at lines 14 to 16 sends a buy order to the server. As no check exists for the minimal distance of stop orders (*StopLoss* and *TakeProfit*), the order will be rejected if *StopLoss* and *TakeProfit* ale closer to the opening price than the minimum distance set by the broker. It is good practice to check for the minimum distance before sending the order request. If the check fails, the solution would be either not sending the request at all and thus saving some processing time or adjusting *StopLoss* and *TakeProfit* to the allowed values.

Then the *globalBalans* environment variable is set to the current account balance at line 18. Afterwards, the *globalPosic* environment variable is increased by 1 in the *globPos()* function and it is set to 0 in next line. Therefore these two lines can be considered useless.

Then result of the order request is checked. The *OrderSend()* function returns the ticket number if the order was successful, otherwise it returns 0. Therefore if the ticket is 0, the order was not successful and an error is printed. If the ticket is greater than 0, the order was successful. Then the check whether the order is among open orders is executed. However, this is not necessary.

Listing 2.7

```
1  int OpenLong(double volume=0.1)
2  // the function opens a long position with lot size=volume
3  {
4      int slippage=10;
5      string comment="20/200 expert v2 (Long)";
6      color arrow_color=Red;
7      int magic=0;
8
9      if (GlobalVariableGet("globalBalans")>AccountBalance())
10         volume=lot*BigLotSize;
11     // if (GlobalVariableGet("globalBalans")>AccountBalance())
```

```
12        //        if (AutoLot) LotSize();
13
14        ticket=OrderSend(Symbol(),OP_BUY,volume,Ask,slippage,
15            Ask-StopLoss_L*Point,Ask+TakeProfit_L*Point,comment,
16            magic,0,arrow_color);
17
18        GlobalVariableSet("globalBalans",AccountBalance());
19        globPos();
20        // if (GlobalVariableGet("globalPosic")>25)
21        // {
22        GlobalVariableSet("globalPosic",0);
23        // }
24
25        if(ticket>0)
26        {
27            if(OrderSelect(ticket,SELECT_BY_TICKET,MODE_TRADES))
28            {
29                return(0);
30            }
31            else
32            {
33                Print("OpenLong(),OrderSelect() - returned
34                    an error : ",GetLastError());
35                return(-1);
36            }
37        }
38        else
39        {
40            Print("Error opening Buy order : ",GetLastError());
41            return(-1);
42        }
43    }
```

OpenShort()

This function works analogically to *OpenLong()*. The only difference is in the statement at lines 12 to 14 in Listing 2.8, which creates a sell market order instead of buy market order.

Listing 2.8

```
1   int OpenShort(double volume=0.1)
2   // The function opens a short position with lot size=volume
3   {
4       int slippage=10;
5       string comment="Gabriel Eze Junior >>>SHORT";
6       color arrow_color=Red;
7       int magic=0;
8
9       if (GlobalVariableGet("globalBalans")>AccountBalance())
10          volume=lot*BigLotSize;
11
12      ticket=OrderSend(Symbol(),OP_SELL,volume,Bid,slippage,
```

```
13              Bid+StopLoss_S*Point,Bid-TakeProfit_S*Point,comment,
14              magic,0,arrow_color);
15          GlobalVariableSet("globalBalans",AccountBalance());
16          globPos();
17          // if (GlobalVariableGet("globalPosic")>25)
18          // {
19          GlobalVariableSet("globalPosic",0);
20          // }
21
22          if(ticket>0)
23          {
24              if(OrderSelect(ticket,SELECT_BY_TICKET,MODE_TRADES))
25              {
26                  return(0);
27              }
28              else
29              {
30                  Print("OpenShort(),OrderSelect() - returned
31                      an error : ",GetLastError());
32                  return(-1);
33              }
34          }
35          else
36          {
37              Print("Error opening Sell order : ",GetLastError());
38              return(-1);
39          }
40      }
```

init()

As mentioned before, the *init()* function is run only once. In case of this EA the environment variable *globalBalans* and the useless *globalPosic* are initialized here. In other words, if they do not already exist, they are created and set to the default value.

Listing 2.9

```
1   int init()
2   {
3       // control of a variable before using
4       if(!GlobalVariableCheck("globalBalans"))
5           GlobalVariableSet("globalBalans",AccountBalance());
6       if(!GlobalVariableCheck("globalPosic"))
7           GlobalVariableSet("globalPosic",0);
8       return(0);
9   }
```

deinit()

The *deinit()* function does nothing in this EA and it could have been omitted altogether.

Listing 2.10

```
1  int deinit()
2  {
3      return(0);
4  }
```

start()

As mentioned before, the *start()* function is executed every time when a new tick is received as long as the EA is in the tick-awaiting mode, i.e. when it is initialized and not in execution of a previous tick.

First, at line 3 in Listing 2.11 it checks if the current hour is later than the one set in the *TradeTime* parameter. If it is the case, the *cantrade* variable is set to *true*.

With the *OrdersTotal()* function the number of open and pending orders is obtained. If the number of open orders is less than the number of the maximum allowed open orders set by the input parameter *Orders*, then a new position might be opened.

At line 10 it is checked that the current hour equals the one set in the *TradeTime* input variable and whether a trade has already been executed at this hour. In other words, this condition equals to *true* only if the current hour is the desired *TradeTime* and that no trade has been executed in this hour and day thus far.

In the block between line 13 and line 28 the trend is determined by subtracting the open values at bars *t1* and *t2* before the current bar. If the difference is greater than the value denoted by *delta_S* multiplied by *Point*, the trade should be executed. However, it is necessary to check if there is enough money on the account. This is done by the *AccountFreeMarginCheck()* part of the condition. The *GetLastError()* part is useless here, as the error 134 is NOT_ENOUGH_MONEY_ERROR and, with the *AccountFreeMarginCheck()* check and the given structure of the EA, it would never occur.

If the account has sufficient funds, the *OpenShort()* function is called to execute the trade. Then *cantrade* is set to *false*, therefore no more trades will be open during this hour. However, the small error here is the lack of the *OpenShort()* error handling. If the request for opening a short position is rejected, *cantrade* will be set to *false* anyway, resulting in no trade open in the given trading hour. Then *return* is called.

The block of code between line 30 and line 44 contains logic for deciding whether or not to open a long postion. It works analogically to the previous one.

If *MaxOpenTime* is set to something else than 0, the block between line 52 and line 81 loops through all open orders and if any of them is open for a longer period of time than the number of hours set in the *MaxOpenTime* input variable, it is closed.

There is one peculiarity with this block. If a trade is executed in the previous blocks, the control never gets to this one. Therefore trades might be closed one tick later. In case of this EA it is insignificant as it works on the one-hour timeframe. Regardless, it is not considered a good practice.

Listing 2.11

```
1  int start()
2  {
3      if((TimeHour(TimeCurrent())>TradeTime)) cantrade=true;
4      // check if there are open orders ...
5      total=OrdersTotal();
6      if(total<Orders)
7      {
8          // ... if no open orders, go further
9          // check if it's time for trade
10         if((TimeHour(TimeCurrent())==TradeTime)&&(cantrade))
11         {
12             // ... if it is
13             if (((Open[t1]-Open[t2])>delta_S*Point)) //if it is
14             {
15                 //condition is fulfilled, enter a short
16                 // position: check if there is free money
17                 // for opening a short position
18                 if(AccountFreeMarginCheck(Symbol(),OP_SELL,
19                     lot)<=0 || GetLastError()==134)
20                 {
21                     Print("Not enough money");
22                     return(0);
23                 }
24                 OpenShort(lot);
25                 //prohibit repeated trade until the next bar
26                 cantrade=false;
27                 return(0);
28             }
29             //if the price increased by delta
30             if (((Open[t2]-Open[t1])>delta_L*Point))
31             {
32                 // condition is fulfilled, enter a long
33                 // position: check if there is free money
34                 if(AccountFreeMarginCheck(Symbol(),OP_BUY,
35                     lot)<=0 || GetLastError()==134)
36                 {
37                     Print("Not enough money");
38                     return(0);
39                 }
40                 OpenLong(lot);
41
42                 cantrade=false;
43                 return(0);
44             }
45         }
46     }
47
```

```
48      // block of a trade validity time checking,
49      // if MaxOpenTime=0, do not check.
50      if(MaxOpenTime>0)
51      {
52          for(cnt=0;cnt<total;cnt++)
53          {
54              if (OrderSelect(cnt, SELECT_BY_POS, MODE_TRADES))
55              {
56                  tmp = (TimeCurrent()-OrderOpenTime())/3600.0;
57                  if (((NormalizeDouble(tmp,8)-MaxOpenTime)>=0))
58                  {
59                      RefreshRates();
60                      if (OrderType()==OP_BUY)
61                          closeprice=Bid;
62                      else
63                          closeprice=Ask;
64                      if (OrderClose(OrderTicket(),OrderLots(),
65                          closeprice,10,Green))
66                      {
67                          Print("Forced closing of the
68                              trade - ", OrderTicket());
69                          OrderPrint();
70                      }
71                      else
72                          Print("OrderClose() in block of a
73                          trade validity time checking returned
74                          an error - ",GetLastError());
75                  }
76              }
77              else
78                  Print("OrderSelect() in block of a trade
79                      validity time checking returned
80                      an error - ",GetLastError());
81          }
82      }
83      return(0);
84  }
```

Summary

This EA opens a trade once per day at a given hour if a trend condition is satisfied. It sets *StopLoss* and *TakeProfit* for each trade. It also allows the trader to set the maximum time for the trade being open.

This EA was probably not written by a programmer as it is poorly written and has some basic logical mistakes in the control flow. Also, it uses deprecated functions (*init()*, *start()*, *deinit()*) from the time before MetaTrader build 600.

2.4.2 Genie

Another algorithm to be analyzed is Genie. It is a slightly advanced algorithm which is building on data from Parabolic SAR indicator created by Genaro

Gravoso. Unlike Geedo, it already has all the neccessary input parameters enabled. Therefore, no changes are needed.

Head

The head of the algorithm already suggests that it is of a higher quality than the previous one. It declares `#property strict`, which enables the most C++-like compiler functionality. This parameter breaks the backward compatibility with the pre-build-600 EAs. However, as this algorithm is programmed in a post-build-600 style, it is insignificant.

Another positive feature is the definition of the magic number in the line `#define MAGICMA 20140730`. All the trades manipulated by this EA will bear this magic number. In this way the EA can recognize trade orders not belonging to it. Thus these orders are not affected by the EA. All of this results in a possibility of concurrent manual and automatic trading in one currency pair window.

The input parameters in Listing 2.12 are declared in the head as well.

Listing 2.12: Genie input parameters

```
1  input double TakeProfit     =500;
2  input double Lots           =0.01;
3  input double TrailingStop   =200;
4  input double MaximumRisk    =0.02;
5  input double DecreaseFactor=3;
6  input double Step           =0.02;    // Acceleration Factor
```

CalculateCurrentOrders()

This function loops through all the open orders from which it chooses only those belonging to the EA (based on the magic number) and counts the number of buy and sell orders. If any buy orders exist, it returns their count, otherwise the number of sell orders is returned as a negative number.

When this function is called later in the code, its return value is only checked against zero (i.e. the case when no trades are open). Therefore, it could have been simplified by returning only *true* if a trade is open or *false* otherwise. It could have been renamed to *IsTradeOpen()*. In this way, looping over all the orders could have been stopped at the first open order. It would make the function run faster. All of this would improve the readability of this EA while not changing its functionality at all.

Listing 2.13

```
1  int CalculateCurrentOrders(string symbol)
2  {
3      int buys=0,sells=0;
4      //---
5      for(int i=0;i<OrdersTotal();i++)
```

```
6       {
7           if(OrderSelect(i,SELECT_BY_POS,MODE_TRADES)==false)
8               break;
9           if(OrderSymbol()==Symbol() &&
10              OrderMagicNumber()==MAGICMA)
11          {
12              if(OrderType()==OP_BUY)   buys++;
13              if(OrderType()==OP_SELL)  sells++;
14          }
15      }
16      //--- return orders volume
17      if(buys>0) return(buys);
18      else       return(-sells);
19  }
```

LotsOptimized()

The optimal lot size is determined by this function. At line 3 in Listing 2.14 the local variable *lot* is defined and initialized to the value of the input parameter *Lots*. However, this initialization is useless, as the *lot* variable is overwritten at line 8, resulting in the value of *Lots* not being used.

The statement at line 4 loads all the past trades. "The history list size depends on the current settings of the *Account history* tab of the terminal." [16, Ch. OrdersHistoryTotal] A small issue with this line is that the *HistoryTotal()* function is deprecated and *OrderHistoryTotal()* should be used instead.

The initialization of the local variable for the counting of losing trades in the trades history is at line 5.

Line 8 sets *lot* to have value of account free margin multiplied by *MaximumRisk*. The interval of allowed values for *MaximumRisk* is (0, 1> where 0 is 0% and 1 means 100%. The value is then divided by 1 000. This suggests that author expects leverage to be 1:100. Therefore, this EA might not work well on accounts with other levels leverage. Subsequently, the value is normalized to one decimal place and assigned to *lot*.

If the *DecreaseFactor* variable is set to a value different from 0, the algorithm loops through all orders in the history starting with the most recent one and counts the number of consecutive losing ones (lines 13 to 25). As soon as it finds a profitable trade, it stops. If there had been more than one losing trade, the lot size is decreased by $lot \times losses/DecreaseFactor$.

If *lot* is too small, set it to 0.1 (line 30). Then return the lot size.

Listing 2.14

```
1   double LotsOptimized()
2   {
3       double lot=Lots;
4       int    orders=HistoryTotal();    // history orders total
5       int    losses=0;                 // number of losses
6                                        //  orders without a break
7       //--- select lot size
```

```
 8       lot=NormalizeDouble(AccountFreeMargin()*MaximumRisk/1000.0,
 9           1);
10       //--- calcuulate number of losses orders without a break
11       if(DecreaseFactor>0)
12       {
13           for(int i=orders-1;i>=0;i--)
14           {
15               if(OrderSelect(i,SELECT_BY_POS,MODE_HISTORY)==false)
16               {
17                   Print("Error in history!");
18                   break;
19               }
20               if(OrderSymbol()!=Symbol() || OrderType()>OP_SELL)
21                   continue;
22               //---
23               if(OrderProfit()>0) break;
24               if(OrderProfit()<0) losses++;
25           }
26           if(losses>1)
27           lot=NormalizeDouble(lot-lot*losses/DecreaseFactor,1);
28       }
29       //--- return lot size
30       if(lot<0.1) lot=0.1;
31       return(lot);
32   }
```

CheckForOpen()

Line 3 in Listing 2.15 declares the local variables which will be used to hold values from the market indicastors. At line 4 there is a declaration of two more local variables. *Ticket* will be used to hold the current trade number or 0 if the opening of a trade position using *OrderSend()* failed, and *total* will contain the count of open orders.

Line 6 ensures that the trading will only be executed during the first tick of a new bar. `Volume[0]` gives the number of ticks at the given bar, with 0 determining the latest bar.

The block of lines 9 to 15 retrieves the values from the market indicators, namely the SAR and the ADX indicator. SAR stands for Parabolic Stop and Reverse system indicator, while ADX is the Average Directional Movement Index indicator. For more information about these indicators see [16, Ch. iSAR] and [16, Ch. iADX] respectively. Notice that the SAR indicator uses an input variable *Step*. This variable is usually set to 0.02. For optimization purposes the range between 0.01 and 0.03 will be tested.

Line 17 initializes the local variable *total* to the number of open orders.

The rest of the function is only executed when the number of total open orders is less than three. It is not clear what the developer's intention was using this condition. The *OnTick()* function ensures that *CheckForOpen()* is called only when no sell nor buy orders are open by the EA. Therefore, the number of

orders returned from *OrdersTotal()* were opened manually or by some other EA. To me, the most sensible explanation is that this condition checks the amount of manual trading that currently takes place and, based on that, this EA either stops opening new trades (if the number of manually opened trades is three or more) or it trades simultaneously with manual trading (if the number of manually opened positions is one or two).

Given that the number of manually opened trades is less than three, this function continues with the block of lines 21 to 26. These lines check if the available free margin is at the level counted by $1000 \times Lots$, where *Lots* is the input variable. If the free margin is lower than this minumum required value, then the function is terminated.

The block of code between line 28 and 45 checks the level of all market indicators and, based on them, it decides whether it is a suitable time to open a sell order. If so, it sends the SELL order using *OrderSend()* of a size determined by the function *LotsOptimized()*, with allowed *slippage* of three points, *stoploss* not set (set to 0), *takeprofit* set using the *TakeProfit* input variable, and the *magic number* to identify that the trade was created by this EA. If opening of the position was successful, then *ticket* will contain the ticket number. In case of failure the ticket will have a value of 0. Depending on the result of the opening of the order, either `("SELL order opened: ",OrderOpenPrice())` or an error will be printed.

The block between line 47 and 64 works analogically to the previous one, just for opening a buy order instead of a sell order.

In both cases (opening of a sell and a buy order) not setting *stoploss* is a huge risk and I advise against it, even though it will be set in the following call of *CheckForClose()*. However, as this EA trades only for the first tick of a new bar, *CheckForClose()* will be closed at the formation of the next bar which can be, depending on the timeframe, in one minute or one day. A lot can change during that time. That is why I consider not setting *stoploss* immediately to be causing a considerable risk.

Listing 2.15

```
1  void CheckForOpen()
2  {
3      double Previous,Current,PDI1,PDI,MDI1,MDI,ADX;
4      int    ticket,total;
5      //--- go trading only for first tiks of new bar
6      if(Volume[0]>1) return;
7      //--- to simplify the coding and speed up access data are
8      // put into internal variables
9      Current=iSAR(NULL,0,Step,0.2,0);
10     Previous=iSAR(NULL,0,Step,0.2,1);
11     ADX=iADX(NULL,0,14,PRICE_CLOSE,MODE_MAIN,0);
12     PDI=iADX(NULL,0,14,PRICE_CLOSE,MODE_PLUSDI,0);
13     PDI1=iADX(NULL,0,14,PRICE_CLOSE,MODE_PLUSDI,1);
14     MDI=iADX(NULL,0,14,PRICE_CLOSE,MODE_MINUSDI,0);
15     MDI1=iADX(NULL,0,14,PRICE_CLOSE,MODE_MINUSDI,1);
```

```
16
17      total=OrdersTotal();
18      if(total<3)
19      {
20          //--- no opened orders identified
21          if(AccountFreeMargin()<(1000*Lots))
22          {
23              Print("We have no money. Free Margin = ",
24                  AccountFreeMargin());
25              return;
26          }
27          //--- sell conditions
28          if(Previous<Close[1] && Current>Close[0] &&
29              PDI1>MDI1 && PDI<MDI && ADX>PDI && ADX>MDI)
30          {
31              ticket=OrderSend(Symbol(),OP_SELL,LotsOptimized(),
32                  Bid,3,0,Bid-TakeProfit*Point,"Genie",
33                  MAGICMA,0,Red);
34              if(ticket>0)
35              {
36                  if(OrderSelect(ticket,SELECT_BY_TICKET,
37                      MODE_TRADES))
38                      Print("SELL order opened: ",
39                          OrderOpenPrice());
40              }
41              else
42                  Print("Error opening SELL order: ",
43                      GetLastError());
44              return;
45          }
46          //--- buy conditions
47          if(Previous>Close[1] && Current<Close[0] &&
48              PDI1<MDI1 && PDI>MDI && ADX>PDI && ADX>MDI)
49          {
50              ticket=OrderSend(Symbol(),OP_BUY,LotsOptimized(),
51                  Ask,3,0,Ask+TakeProfit*Point,"Genie",
52                  MAGICMA,0,Blue);
53              if(ticket>0)
54              {
55                  if(OrderSelect(ticket,SELECT_BY_TICKET,
56                      MODE_TRADES))
57                      Print("BUY order opened: ",
58                          OrderOpenPrice());
59              }
60              else
61                  Print("Error opening BUY order: ",
62                      GetLastError());
63              return;
64          }
65          return;
66      }
67      //---
68  }
```

CheckForClose()

This function checks if some of the orders opened by this EA can be closed or should be modified when it comes to the trailing stop.

Again, line 4 in Listing 2.16 ensures that the trading will only be executed during the first tick of a new bar.

The block between line 14 and 34 is executed if the current order is of the buy type. First, the current *stoploss* value is checked, and if its distance from the current price is greater than that set by the trailing stop, or if it is not set, then the order is updated with a new *stoploss* value and the function is finished.

If it is not necessary to adjust *stoploss*, the next condition checks whether the order should be closed or not. The `Bid-OrderOpenPrice()>Ask-Bid` part of the condition checks if the trade is profitable. The `Open[1]>Close[1]` part of the condition checks if the market went down in the last bar. So, if the market went down and despite that we are still profitable, the order will be closed.

Analogically, the block of code between line 35 and 55 works for a sell order.

Listing 2.16

```
1  void CheckForClose()
2  {
3      //--- go trading only for first tiks of new bar
4      if(Volume[0]>1) return;
5      //---
6      for(int i=0;i<OrdersTotal();i++)
7      {
8          if(OrderSelect(i,SELECT_BY_POS,MODE_TRADES)==false)
9              break;
10         if(OrderMagicNumber()!=MAGICMA ||
11             OrderSymbol()!=Symbol())
12             continue;
13         //--- check order type
14         if(OrderType()==OP_BUY)
15         {
16             if(OrderStopLoss()<Bid-Point*TrailingStop ||
17                 OrderStopLoss()==0)
18             {
19                 //--- modify order and exit
20                 if(!OrderModify(OrderTicket(),OrderOpenPrice(),
21                     Bid-Point*TrailingStop,OrderTakeProfit(),
22                     0,Green))
23                     Print("OrderModify error ",GetLastError());
24                 return;
25             }
26             if(Bid-OrderOpenPrice()>Ask-Bid &&
27                 Open[1]>Close[1])
28             {
29                 if(!OrderClose(OrderTicket(),OrderLots(),
```

```
30                          Bid,1,White))
31                          Print("OrderClose error ",GetLastError());
32                  }
33                  break;
34              }
35              if(OrderType()==OP_SELL)
36              {
37                  if(OrderStopLoss()>Ask+Point*TrailingStop ||
38                      OrderStopLoss()==0)
39                  {
40                      //--- modify order and exit
41                      if(!OrderModify(OrderTicket(),OrderOpenPrice(),
42                          Ask+Point*TrailingStop,OrderTakeProfit(),
43                          0,Red))
44                          Print("OrderModify error ",GetLastError());
45                      return;
46                  }
47                  if(OrderOpenPrice()-Ask>Ask-Bid &&
48                      Open[1]<Close[1])
49                  {
50                      if(!OrderClose(OrderTicket(),OrderLots(),
51                          Ask,1,White))
52                          Print("OrderClose error ",GetLastError());
53                  }
54                  break;
55              }
56          }
57      }
```

OnTick()

OnTick(), the equivalent of the deprecated *start()* function, is quite simple. The lines 4 to 5 in Listing 2.17 check if the EA is allowed to trade and if sufficient amount of bars is loaded in history. This is necessary for the proper functioning of the market indicators. At lines 7 to 10 it is checked that no orders are opened by this EA. If so, it opens one if the conditions are fulfilled. If some orders are already open, it either updates their *stoploss* level, if necessary, or closes them, if they can be closed.

Listing 2.17

```
1   void OnTick()
2   {
3       //--- check for history and trading
4       if(Bars<100 || IsTradeAllowed()==false)
5           return;
6       //--- calculate open orders by current symbol
7       if(CalculateCurrentOrders(Symbol())==0)
8           CheckForOpen();
9       else
10          CheckForClose();
11      //---
```

```
12  }
```

Summary

The Genie Expert Advisor can be evaluated positively with regard to the programming style. It uses the newest MQL4 standards (except for one use of the deprecated *HistoryTotal()* function). The code is fairly well-readable. However, *CalculateCurrentOrders()* could have been simplified. Furthermore, the limiting functionality of the EA – when at least three manually opened position are present in the *CheckForOpen()* function – is undocumented and rather confusing. A positive aspect is that it uses the magic number. Additionally, it uses built-in market indicators.

All in all, it is an interesting algorithm and, while not being perfect, it is better written and more advanced than the aforementioned Geedo.

Chapter 3

Implementation

After getting acquainted with the theory and after the analysis of the publicly available Expert Advisors the implementation of a sample Expert Advisor follows. I decided that one of the features of my Expert Advisor should be a consistent structure, so that this algorithm can be easily reused as a template for new algorithms in the future.

Another goal is the creation of a new trading strategy. The focus will be on a criterion for opening a trade, while a criterion for closing a trade will be a simple stop-loss (trailing stop) and take-profit. Also, a smart function for determining the trad esize will be implemented.

For the sake of simplicity, only one position of the buy type and one position of the sell type can be open at the time.

3.1 Trade Opening Criterion

The trading criterion of my EA is based on two simple moving averages – one with a short period (magnitude of tens of bars) and one with a medium/long period (magnitude of hundreds of bars). From these moving averages the *short-term* and the *midterm trend* are determined.

The EA looks for abrupt changes of the market in a direction outwards from the midterm trend. If this abrupt change slows down, the EA expects that the market will return back towards the midterm trend. A buy market order is sent in case this abrupt deviation from the midterm trend happens in a downward direction. If the abrupt deviation from the midterm trend happens in an upward direction, a sell order is opened. See Figure 3.1.

In order to be able to program the behavior described in the previous paragraph, it is necessary to break it down to a few basic steps. The three main steps are:

1. Checking for an abrupt deviation in a direction outwards from the midterm trend.

2. After the completion of step 1, wait for the end (slow down) of this abrupt change in order to open a position.

3. Simple check of the wait duration in step 2 and terminating it if this duration is longer than a certain threshold.

Figure 3.1: My Expert Advisor trading style explanation

In Figure 3.1 the yellow line represents the midterm simple moving average. The red line is the short-term simple moving average and the silver lines show its threshold.

Step 1 Detailed

I have defined a set of conditions which must be fulfilled for the current market state to be considered an abrupt deviation outward from the midterm trend.

For a downward abrupt deviation the conditions are the following:

1. The midterm trend must be ranging or upwards.

2. The current ask price must be lower than the short-term moving average (minus a certain threshold).

3. The short-term moving average must be lower than the midterm moving average.

For an upward abrupt deviation the conditions are the following:

1. The midterm trend must be ranging or downwards.
2. The current bid price must be higher than the short-term moving average (plus a certain threshold).
3. The short-term moving average must be higher than the midterm moving average.

Step 2 Detailed

In order to determine whether the abrupt deviation from the midterm trend stopped or slowed down, it is necessary to check the short-term trend. For a downward abrupt deviation slowdown the condition is that the short-term trend is ranging or having an upward direction. For an upward abrupt deviation slowdown the condition is that the short-term trend is ranging or having a downward direction.

Remarks

The algorithm trades only the first tick of a new bar. Therefore, the most relevant value, on which the moving average is calculated, is the close price of the previous bar.

As it is difficult to guess the correct values for constants such as the threshold, I have decided to change most of the hardcoded constants into input parameters. In this way, the optimal values can be found using optimization.

The trader can set the period of both the short-term and the midterm simple moving average. Then the threshold from the short-term simple moving average (used in Step 1: Condition 2) can be set as well.

Additionally, for determining the trend of the simple moving average the number of bars can be set. Also, the tangent of the angle can be set to set the limit between the trending and the ranging moving average. This can be done for both the short-term and the midterm simple moving average trends.

Another available input parameter is the maximum duration of waiting for an abrupt deviation from the midterm simple moving average slowdown used in Step 3.

3.2 Trade Closing Criterion

As was already mentioned, the trade closing criterion is fairly simple. It consists of setting the stop-loss, the trailing stop and the take-profit values. However, again, it is difficult to determine permanent values for these values, as the market is changing all the time. That is why the stop-loss, the trailing stop and the take-profit levels were made available as input parameters and thus can be used for optimization.

3.3 Trade Size Determination

As for the trade size, I gave the user a lot of freedom. The trader can either set the trade size to a constant amount of lots or it can be set to a percentage of the free margin available for the account.

3.4 Physical Implementation

In the following sections the functions of my Expert Advisor will be explained one by one. It was built based on the guidelines from the official guide [21, Ch. Creation of a Normal Program]. However, these guidelines are slightly outdated as they still use the old pre-build600 constructs. Therefore, I decided to improve them by using contemporary language constructs.

3.4.1 Head

Listing 3.1

```
1  //+------------------------------------------------------------------+
2  //|                                                       MyAlg.mq4  |
3  //|                                                    Juraj Korcek  |
4  //|                                                                  |
5  //+------------------------------------------------------------------+
6  #property copyright "Juraj Korcek"
7  #property link      ""
8  #property version   "1.00"
9  #property strict
10
11 //+------------------------------------------------------------------+
12 //| Includes                                                         |
13 //+------------------------------------------------------------------+
14 #include <MarketProperties.mqh>
15 #include <OrdersBook.mqh>
16
17 //+------------------------------------------------------------------+
18 //| Input parameters                                                 |
19 //+------------------------------------------------------------------+
20 input int       i_magicNumber = 1231564651;
21                  // Magic number
22
23 input double    i_relativeLotSize = 0.1;
24                  // Lot size as a percentage of
25                  //  Free Margin (0; 100>
26 input double    i_lots = 0;
27                  // Size of order; if 0, lot size is
28                  // caluclated using relativeLotSize
29
30 input int       i_MAshortTermPeriod = 10;
31                  // Period of Shorterm MA
32 input int       i_MAshortTermThreshold = 50;
```

```
33                    // Threshold of Shorterm MA
34  input int         i_MAshortTermTrendBars = 1;
35                    // Number of bars to determine the
36                    // trend of Shortterm MA
37  input double      i_MAshortTermTrendToleranceAngleTan = 0.4;
38                    // Tangens of limit angle when Shortterm
39                    // MA trend is considered to be ranging
40  input int         i_MAmidTermPeriod = 250;
41                    // Period of Midterm MA
42  input int         i_MAmidTermTrendBars = 10;
43                    // Number of bars to determine the trend
44                    // of Midterm MA
45  input double      i_MAmidTermTrendToleranceAngleTan = 0.4;
46                    // Tangens of limit angle when Midterm MA
47                    // trend is considered to be ranging
48  input int         i_flagTimer = 250;
49                    // Number of ticks for buy or sell flag
50                    // stays triggered
51
52  input int         i_stopLoss = 50;
53                    // StopLoss in points
54  input int         i_takeProfit = 200;
55                    // TakeProfit in points
56  input int         i_trailingStop = 100;
57                    // TrailingStop in points
58
59  //+------------------------------------------------------------------+
60  //| Global variables                                                 |
61  //+------------------------------------------------------------------+
62  bool g_buy_flag = False;
63  bool g_sell_flag = False;
64  int g_timer_buy_flag = 0;
65  int g_timer_sell_flag = 0;
66
67  MarketProperties* g_marketProperties;
68  OrdersBook* g_ordersBook;
```

The head of the EA in Listing 3.1 is split into several distinct parts.

First, the basic properties are set. The interesting one is `#property strict`, which ensures the compiler behavior to be as close as possible to the one of the standard C++ compiler, although it breaks the backward compatibility with the terminals of an earlier build than 600.

Then, the includes are defined starting on line 14. These lines include classes which make bookkeeping and holding of the market information easier.

The declaration and the initialization of the input variables follows. First, the magic number is set. It will be used to distinguish trades opened by this EA from other trades. Then, at lines 23 to 28 the input parameters affecting the order size are set. The next group of parameters is the one used in tweaking of the trading criterion. The last group of input parameters sets the limits for closing a trade.

The last part (starting at line 62) of the head is used for the declaration and the initialization of the global variables.

3.4.2 OnInit()

Listing 3.2

```
1  //+------------------------------------------------------------------+
2  //| Expert initialization function                                   |
3  //+------------------------------------------------------------------+
4  int OnInit()
5  {
6      g_marketProperties = new MarketProperties();
7      Terminal();
8      return(INIT_SUCCEEDED);
9  }
```

During the initialization of the EA, the current market properties are assigned to the global variable *g_marketProperties*. Then the function *Terminal()* is called for the creation of the orders book. For more information on *Terminal()* function see Section 3.4.5.

3.4.3 OnDeinit()

Listing 3.3

```
1  //+------------------------------------------------------------------+
2  //| Expert deinitialization function                                 |
3  //+------------------------------------------------------------------+
4  void OnDeinit(const int reason)
5  {
6      delete g_marketProperties;
7      delete g_ordersBook;
8  }
```

The only role of the deinitialization in my EA is the cleanup of allocated objects.

3.4.4 OnTick()

Listing 3.4

```
1  //+------------------------------------------------------------------+
2  //| Expert tick function                                             |
3  //+------------------------------------------------------------------+
4  void OnTick()
5  {
6      PlaySound("tick.wav");
7      Terminal();
8      Events();
9      Trade(Criterion());
10 }
```

As a rule, *OnTick()* is the main function of the EA. This case is no exception. At line 6 a sound is played to notify the user about a new tick. This might become annoying after some time. It is used to demonstrate the option of playing sounds from an EA. Then the orders book is updated using the *Terminal()* function and the market info is updated using the *Events()* function. For more information on *Events()* see Section 3.4.6. Finally, the trade criterion is checked in the *Criterion()* function and, based on its result, the *Trade()* function either executes a trade or not. For more information on *Criterion()* and *Trade()* functions see Section 3.4.8 and Section 3.4.10 respectively.

3.4.5 Terminal()

Listing 3.5

```
1  //+------------------------------------------------------------------+
2  //| (user-defined) Function for keeping book of orders               |
3  //+------------------------------------------------------------------+
4  void Terminal()
5  {
6      int qnt = 0;
7
8      delete g_ordersBook;
9      g_ordersBook = new OrdersBook();
10
11     for(int i=0; i<OrdersTotal(); i++)
12     {
13         if((OrderSelect(i, SELECT_BY_POS) == true) &&
14             (OrderSymbol() == Symbol()) &&
15             OrderMagicNumber() == i_magicNumber)
16         {
17             g_ordersBook.Orders[qnt] = new Order(
18                                             OrderOpenPrice(),
19                                             OrderStopLoss(),
20                                             OrderTakeProfit(),
21                                             OrderTicket(),
22                                             OrderLots(),
23                                             OrderType(),
24                                             OrderMagicNumber());
25             g_ordersBook.OrderTypeCount[OrderType()]++;
26             qnt++;
27         }
28     }
29     g_ordersBook.OrderCount = qnt;
30 }
```

This function initializes the orders book. It iterates through all the orders and picks only those with a desired symbol (currency pair) and a desired magic number, i.e. orders belonging to this EA. The filtered order is saved into the *Orders* array with the information about its open price, stop-loss, take-profit, ticket number, size and type. Additionally, the count of each order type is kept

in the *OrderTypeCount* array. In the end, the count of all orders belonging to this EA is saved into the *OrderCount* member variable of *g_ordersBook*.

3.4.6 Events()

Listing 3.6

```
1  //+------------------------------------------------------------------+
2  //| (user-defined) Function for determing interesting                |
3  //| server changes                                                   |
4  //+------------------------------------------------------------------+
5  void Events()
6  {
7      g_marketProperties.Update();
8      if(g_marketProperties.m_minimumDistance.Changed() == true)
9          Print("New minimum distance: " +
10                 g_marketProperties.m_minimumDistance.Get());
11 }
```

In Listing 3.6 the code of the *Events()* function is shown. This function is responsible for maintaining up-to-date information about the server – in particular, about the minimum distance set by the broker, one lot cost, the minimum allowed lot size and the lot step. As usually, only the minimum distance is changed by the broker during the day (often during high market volatility, e.g. at a news announcement), the function checks for a change of this option and logs a new value in case a change occurred.

3.4.7 Lot()

Listing 3.7

```
1  //+------------------------------------------------------------------+
2  //| (user-defined) Function for determing order size                 |
3  //+------------------------------------------------------------------+
4  double Lot()
5  {
6      double oneLotCost = g_marketProperties.m_oneLotCost.Get();
7      double minLot = g_marketProperties.m_minimumLotSize.Get();
8      double lotStep = g_marketProperties.m_lotStep.Get();
9      double freeMargin = AccountFreeMargin();
10     double relativeLotSize;
11     double orderSize;
12
13     if(i_lots > 0)
14     {
15         double moneyRequired = i_lots * oneLotCost;
16         if(moneyRequired <= freeMargin)
17             orderSize = i_lots;
18         else
19             orderSize = MathFloor(freeMargin / oneLotCost
20                 / lotStep) * lotStep;
21     }
```

```
22      else
23      {
24          if(i_relativeLotSize > 100)
25              relativeLotSize = 100;
26          else
27              relativeLotSize = i_relativeLotSize;
28          if(i_relativeLotSize == 0)
29              orderSize = minLot;
30          else
31              orderSize = MathFloor(freeMargin
32                                    * relativeLotSize
33                                    / 100
34                                    / oneLotCost
35                                    / lotStep)
36                          * lotStep;
37      }
38      if(orderSize < minLot)
39          orderSize = minLot;
40      if(orderSize * oneLotCost > freeMargin)
41      {
42          Print("Not enough money for " +
43              DoubleToStr(minLot, 2) + lots");
44          return(0);
45      }
46      return(orderSize);
47  }
```

Listing 3.7 shows the workings of the trade size management. If the *i_lots* input variable is different form zero, then its value is used as an absolute trade size in lots. In case of insufficient funds (insufficient account's free margin), the order size is adjusted to the maximum possible value not exceeding the free margin amount and then the size is adjusted to abide with the lot step decided by the broker. In case the *i_lots* input variable is equal to zero, the relative trade size will be determined based on the *i_relativeLotSize* input parameter. At lines 24 to 29 extreme values are checked and adjusted. The value of *i_relativeLotSize* cannot be higher than one hundred as it would always trigger an error about an insufficient margin, while it cannot be equal to zero because orders with a zero size are neither allowed nor do they make sense. Then the order size is adjusted to comply with the broker's requirement of the lot step and, eventually, the check for the minimal lot size limit is executed. If the minimum lot size is exceeding the free margin, an error is logged and a trade size equal to zero is returned.

3.4.8 Criterion()

Listing 3.8

```
1  //+------------------------------------------------------------------+
2  //| (user-defined) Function for determing whether                    |
3  //| to open trade or not                                             |
```

```
4  //+------------------------------------------------------------------+
5  int Criterion()
6  {
7      if(Volume[0] > 1) return(0);
8
9      if(g_timer_buy_flag > 0) g_timer_buy_flag--;
10     if(g_timer_buy_flag == 0) g_buy_flag = False;
11     if(g_timer_sell_flag > 0) g_timer_sell_flag--;
12     if(g_timer_sell_flag == 0) g_sell_flag = False;
13
14     string symbol = Symbol();
15     int period = Period();
16
17     double MAshortTermCur;
18     double MAshortTermPrev;
19     int MAshortTermTrend;
20     double MAshortTermTopThreshold;
21     double MAshortTermBottomThreshold;
22     double MAmidTermCur;
23     double MAmidTermPrev;
24     int MAmidTermTrend;
25
26     MAshortTermCur = iMA(symbol, period,
27                          i_MAshortTermPeriod, 0, MODE_SMA,
28                          PRICE_CLOSE, 1);
29     MAshortTermPrev = iMA(symbol, period,
30         i_MAshortTermPeriod, 0, MODE_SMA,
31         PRICE_CLOSE, 1 + i_MAshortTermTrendBars);
32     MAshortTermTrend = DetermineTrend(MAshortTermCur,
33         MAshortTermPrev,
34         i_MAshortTermTrendBars,
35         i_MAshortTermTrendToleranceAngleTan);
36     MAshortTermTopThreshold = MAshortTermCur +
37         i_MAshortTermThreshold * Point;
38     MAshortTermBottomThreshold = MAshortTermCur -
39         i_MAshortTermThreshold * Point;
40     MAmidTermCur = iMA(symbol, period,
41         i_MAmidTermPeriod, 0, MODE_SMA, PRICE_CLOSE, 1);
42     MAmidTermPrev = iMA(symbol, period,
43         i_MAmidTermPeriod, 0, MODE_SMA,
44         PRICE_CLOSE, 1 + i_MAmidTermTrendBars);
45     MAmidTermTrend = DetermineTrend(MAmidTermCur,
46         MAmidTermPrev,
47         i_MAmidTermTrendBars,
48         i_MAmidTermTrendToleranceAngleTan);
49
50     if(g_buy_flag == True && MAshortTermTrend >= 0)
51     {
52         g_buy_flag == False;
53         return(10);
54     }
55
56     if(g_sell_flag == True && MAshortTermTrend <= 0)
57     {
```

```
58            g_sell_flag = False;
59            return(20);
60        }
61
62        if((MAmidTermTrend == 1 || MAmidTermTrend == 0) &&
63           Ask < MAshortTermBottomThreshold &&
64           MAshortTermCur < MAmidTermCur)
65        {
66            g_buy_flag = True;
67            g_timer_buy_flag = i_flagTimer;
68            return(0);
69        }
70
71        if((MAmidTermTrend == -1 || MAmidTermTrend == 0) &&
72           Bid > MAshortTermTopThreshold &&
73           MAshortTermCur > MAmidTermCur)
74        {
75            g_sell_flag = True;
76            g_timer_sell_flag = i_flagTimer;
77            return(0);
78        }
79
80        return(0);
81    }
```

This function is responsible for determining the trade criterion based on the conditions described in chapter 3.1. Line 7 should ensure that only the first tick of a new bar is traded. Lines 9 to 12 execute the logic of step 3 of the trade opening criterion. Then at lines 17 to 48 all the necessary variables are initialized, including the moving averages values, thresholds and trends. The logic for step 2 (see Chapter 3.1) at lines 50 to 60. Finally, the logic for step 1 (see Chapter 3.1) is executed at lines 62 to 78. The return values of this function are 10 (signaling that a buy order should be opened), 20 (signaling that a sell order should be opened) and 0 (signaling that no order should be opened). The return value is used as an input parameter for the *Trade()* function (see Section 3.4.10).

3.4.9 DetermineTrend()

Listing 3.9

```
1   //+------------------------------------------------------------------+
2   //| (user-defined) Function for trend check                          |
3   //+------------------------------------------------------------------+
4   int DetermineTrend(double currentValue, double previousValue,
5                      int barsBetweenValues, double thresholdTan)
6   {
7       double diff = currentValue - previousValue;
8       double tangens =  (MathAbs(diff) / Point)
9           / float(barsBetweenValues);
10      if(tangens < thresholdTan)
```

```
11          return 0;
12      else if(diff > 0)
13          return 1;
14      else return -1;
15  }
```

This is a simple function for determining whether the market is ranging or moving in a certain direction (trending). It calculates the tangent value by taking the current and the previous price difference in points as the opposite side, and the number of the bars produced between the current and the previous price as the adjacent side. If the absolute value of the result is lower than the threshold, no trend is recorded and 0 is returned. On the other hand, if the absolute value of the result is higher than the threshold and the value of the result is positive, then the trend is in the upward direction and 1 is returned. If the result is negative, the trend is moving downwards and –1 is returned.

3.4.10 Trade()

Listing 3.10

```
1  //+------------------------------------------------------------------+
2  //| (user-defined) Function for executiong and                       |
3  //| modifying trades                                                 |
4  //+------------------------------------------------------------------+
5  void Trade(int Trad_Oper)
6  {
7      double orderSize = Lot();
8
9      switch(Trad_Oper)
10     {
11         case 10:
12             if(orderSize == 0.0)
13                 return;
14             OpenOrder(0, orderSize);
15             return;
16         case 20:
17             if(orderSize == 0.0)
18                 return;
19             OpenOrder(1, orderSize);
20             return;
21         case 0:
22             AdjustTrailingStop(0);
23             AdjustTrailingStop(1);
24             return;
25     }
26 }
```

The *Trade()* function first determines the order size using the *Lot()* function. Then, based on the value returned from the *Criterion()* function, it opens either a buy position (see line 14) or a sell position (see line 19) given

that the order size is not 0. If *Criterion()* returns 0, no new trades are supposed to be opened and, therefore, only the trailing stop is adjusted. For more information on the *Lot()* function see Section 3.4.7.

3.4.11 OpenOrder()

Listing 3.11

```
1  //+------------------------------------------------------------------+
2  //| (user-defined) Function tha opens an order of                    |
3  //| given type                                                       |
4  //+------------------------------------------------------------------+
5  void OpenOrder(int type, double orderSize)
6  {
7      int ticket;
8      int stopLossInPoints;
9      int takeProfitInPoints;
10     double stopLoss;
11     double takeProfit;
12
13     while(g_ordersBook.OrderTypeCount[type] == 0)
14     {
15         int minimumDistance =
16             g_marketProperties.m_minimumDistance.Get();
17         if(i_stopLoss < minimumDistance)
18             stopLossInPoints = minimumDistance;
19         else
20             stopLossInPoints = i_stopLoss;
21         if(i_takeProfit < minimumDistance)
22             takeProfitInPoints = minimumDistance;
23         else
24             takeProfitInPoints = i_takeProfit;
25
26         if(type > 0)
27             Print("Trying to open order Sell..");
28         else
29             Print("Trying to open order Buy..");
30
31         if(type == 0)
32         {
33             stopLoss = Bid - stopLossInPoints *  Point;
34             takeProfit = Bid + takeProfitInPoints * Point;
35             ticket = OrderSend(Symbol(), 0, orderSize, Ask,
36                                2, stopLoss, takeProfit, "",
37                                i_magicNumber);
38         }
39         if(type == 1)
40         {
41             stopLoss = Ask + stopLossInPoints *  Point;
42             takeProfit = Ask - takeProfitInPoints * Point;
43             ticket = OrderSend(Symbol(), 1, orderSize, Bid,
44                                2, stopLoss, takeProfit, "",
45                                i_magicNumber);
```

```
46              }
47
48              if(ticket < 0)
49              {
50                  if(Errors(GetLastError()) == false)
51                      return;
52              }
53              Terminal();
54              Events();
55          }
56
57          return;
58      }
```

This function opens a market order of a given type and size. It also ensures that the stop-loss and the take-profit levels are compliant with the requirements of the broker. It updates the order book and the market information in the end by calling *Terminal()* and *Events()* respectively.

3.4.12 AdjustTrailingStop()

Listing 3.12

```
1   //+------------------------------------------------------------------+
2   //| (user-defined) Function tha adjusts StopLoss                     |
3   //| based on TrailingStop                                            |
4   //+------------------------------------------------------------------+
5   void AdjustTrailingStop(int type)
6   {
7       int ticket;
8       int trailingStopInPoints;
9       double openPrice;
10      double trailingStop;
11      double stopLoss;
12      double takeProfit;
13      bool toBeModified;
14
15      for(int i = 0; i < g_ordersBook.OrderCount; i++)
16      {
17          if(g_ordersBook.Orders[i].Type() != type)
18              continue;
19          toBeModified = false;
20          openPrice = g_ordersBook.Orders[i].OpenPrice();
21          stopLoss = g_ordersBook.Orders[i].StopLoss();
22          takeProfit = g_ordersBook.Orders[i].TakeProfit();
23          ticket = g_ordersBook.Orders[i].Ticket();
24          int minimumDistance =
25              g_marketProperties.m_minimumDistance.Get();
26          if(i_trailingStop < minimumDistance)
27              trailingStopInPoints = minimumDistance;
28          else
29              trailingStopInPoints = i_trailingStop;
30          trailingStop = trailingStopInPoints * Point;
```

```
31
32          switch(type)
33          {
34              case 0:
35                  if(NormalizeDouble(stopLoss, Digits) <
36                  NormalizeDouble(Bid - trailingStop, Digits))
37                  {
38                      stopLoss= Bid - trailingStop;
39                      toBeModified = true;
40                  }
41                  break;
42              case 1:
43                  if(NormalizeDouble(stopLoss, Digits) >
44                  NormalizeDouble(Ask + trailingStop, Digits) ||
45                  NormalizeDouble(stopLoss, Digits) == 0)
46                  {
47                      stopLoss = Ask + trailingStop;
48                      toBeModified = true;
49                  }
50          }
51          if(toBeModified == false)
52              continue;
53          bool result = OrderModify(ticket, openPrice, stopLoss,
54              takeProfit, 0);
55
56          if(result == false)
57          {
58              if(Errors(GetLastError()) == false)
59                  return;
60              i--;
61          }
62          Terminal();
63          Events();
64      }
65      return;
66  }
```

The *AdjustTrailingStop()* function adjusts the stop-loss levels of all orders of a given type based on the trailing-stop setting while respecting the minimum distance requirement set by the broker.

3.4.13 Errors()

Listing 3.13

```
1  //+------------------------------------------------------------------+
2  //| (user-defined) Error handling function                           |
3  //+------------------------------------------------------------------+
4  bool Errors(int errorCode)
5  {
6      if(errorCode == 0)
7          return(false);
8
```

```
 9       switch(errorCode)
10       {
11           case 129:
12               Print("Wrong price.");
13               RefreshRates();
14               return(true);
15           case 135:
16               Print("Price changed.");
17               RefreshRates();
18               return(true);
19           case 136:
20               Print("No prices. Awaiting a new tick.");
21               while(RefreshRates() == false)
22                   Sleep(1);
23               return(true);
24           case 146:
25               Print("Trading subsystem is busy.");
26               Sleep(500);
27               RefreshRates();
28               return(true);
29
30           case 2:
31               Print("Common error.");
32               return(false);
33           case 5:
34               Print("Old version of the terminal.");
35               return(false);
36           case 64:
37               Print("Account is blocked.");
38               return(false);
39           case 133:
40               Print("Trading is prohibited");
41               return(false);
42           default:
43               Print("Occurred error " + errorCode);
44               return(false);
45       }
46   }
```

Listing 3.13 shows the error handling function. The errors are split into two groups – recoverable and unrecoverable. In case of an unrecoverable error *false* is returned, otherwise *true* is returned. The trader should act according to the result of this function (e.g. stop the EA in case of an unrecoverable error).

3.4.14 OrdersBook and MarketProperties

OrdersBook and *MarketProperties* were implemented as support classes to make the main algorithm cleaner and its development faster. For the implementations of these classes see Appendix C and Appendix D respectively.

Chapter 4

Evaluation

My Expert Advisor (see Chapter 3) and the two previously analyzed publicly available algorithms (discussed in Sections 2.4.1 and 2.4.2) will be trained and tested on historical data over different periods of time. Subsequently, they will be compared to each other.

For each EA, two models will be trained:

1. One with the data starting one week before the given point in time.
2. One with the data starting one month before the given point in time.

28th February 2015 was chosen to be the point when the training ends and the testing starts.

Each model will then be tested on five consecutive weeks – one by one. Eventually, the testing results will be used for a comparison of the given Expert Advisors.

4.1 Historical Data Acquisition

It is necessary to choose the historical data source wisely, as proper data are crucial for the precision of the model training and testing. Sources of historical forex data vary wildly in quality and price. The following paragraphs discuss some of the available options.

a) The trader can obtain certain historical data from his broker using MetaTrader 4's *History Center* (*Tools/History Center* or pressing F2). This option is for free and data are immediately available. However, brokers usually offer only a limited number of days (with a periodicity of one minute) of historical data which is not suitable for the model training and testing.

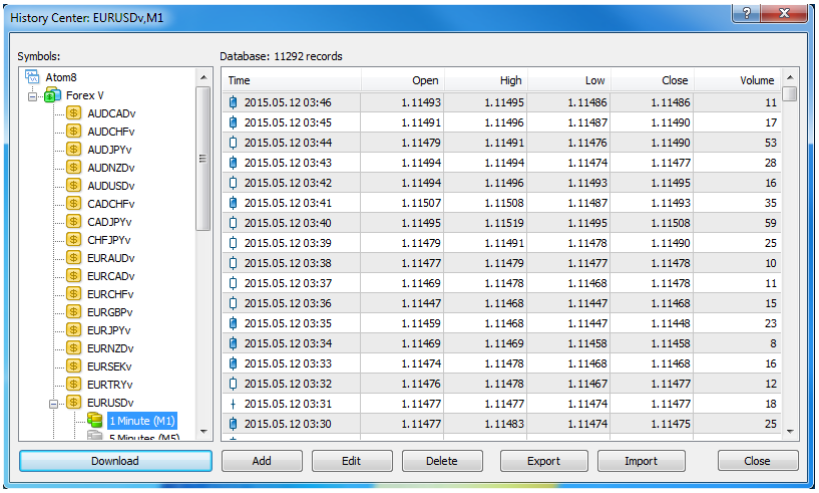

Figure 4.1: History center

b) In case the broker does not offer any historical data in MetaTrader 4's *History Center*, it is possible to download some free data from MetaQuotes Software Corp. (the company developing MetaTrader 4). However, these data also have a span of only a limited number of days with a maximum periodicity of one minute, they often contain holes and the way how MetaQuotes Software Corp. generates these data is unknown. Therefore, this data source is also inappropriate for the model training and testing.

c) Some brokers provide free historical data feeds on their websites. These are quite precise (a periodicity of one tick) and often cover extensive periods of time. However, these feeds are usually not in a format compatible with MetaTrader 4 by default. Therefore, they need to be converted first. For this, the trader might have to write a conversion script himself. Another option is to search the Internet, as there is a good chance that somebody from the community has already developed and shared the required tool.

d) Another option is the buy the historical data. In this case, a vendor is required to deliver the marketed quality and format. This is probably the fastest way to obtain high-precision, wide-timespan historical data, as long as the trader is willing to spend some money.

Atom8, my chosen broker, does not provide long term historical data inside MetaTrader 4's *History Center* nor on their website. Therefore, I have decided to get historical data from a different broker. I chose *Dukascopy SA*, a Swiss forex broker, because it has a reputation of offering high-quality historical data feeds with periodicity of one tick. As Atom8 is a white label partner

broker of Dukascopy [9], there should not be any significant differences in the bid/ask amount levels. In order to prove this, one-day data (open, high, low and close price) were obtained from both brokers and they were compared (see Appendix E).

However, Dukascopy does not offer historical data in a format compatible with MetaTrader 4. For conversion into a compatible format the Tickstory utility can be used. It downloads the data from Dukascopy, converts them into a MetaTrader 4 compatible format and imports them into MetaTrader 4. The following section describes this in detail.

4.1.1 Tickstory Guide

The *Tickstory* utility can be found enclosed on the attached CD. However, to ensure the installation of the newest version it is recommended to download the current version from the official website [23].

1. Install *Tickstory* as any other application. In case of an "access denied" error it is advised to run the installation as the administrator.

2. At the first run, it is necessary to set the application settings.

   a) In the *Dukascopy Datasource* tab test your connection and make changes if required.

   b) In the MetaTrader 4 *Settings* tab fill in the path to your MetaTrader 4 folder. When it comes to the MetaTrader 4 Data Folder, the easiest way how to find it is by opening MetaTrader 4 and selecting *File/Open Data Folder* from the menu.

3. The next step is to select the desired currency pair, right-click it and choose *Export to MT4...*

4. Before exporting the data, it is needed to obtain the configuration of the chosen broker (in my case *Atom8*). In order to generate the configuration file, click the *Help* tab and follow all the steps mentioned there. Note that the procedure will probably not work on weekends and banking holidays. Therefore, if no configuration file is generated, wait until the market is open to repeat this step. The configuration used for testing and training can be found on the enclosed CD.

5. If you finished all the steps mentioned in the *Help* tab (including loading the configuration in the *MetaTrader Info* tab), verify that the symbol in the *MetaTrader Info* tab matches the symbol name used by your broker. Brokers often add some letters after the currency pair name. If the name is the same, you can continue to the next step. However, if the names differ, put the broker's name of the currency pair into the *Map to* field.

6. Change to the *Data export* tab, choose the desired timespan and timeframes, change additional settings if required, and click *OK* in order to export the data.

7. Once the export is finished, check if any errors occurred. If they did, consult the official website for help.

8. If the export ended without errors, run MetaTrader 4 either by clicking the *Launch Metatrader for Back-testing* button or pressing F8.

9. Now, the historical data are ready for the model training and testing.

4.1.2 Historical Data Requirements

The point in time when the training stops and the testing starts is February 28th 2015. Given that the longest training period used will be one month, the historical data has to start on February 1st 2015. Given that the testing period is five weeks long, it is necessary to obtain historical data up to April 5th 2015.

As both Genie and my Expert Advisor use a one-minute timeframe (M1) for trading and Geedo uses a one-hour timeframe (H1), obtaining these two timeframes is sufficient for our needs.

As the Atom8 broker uses *EURUSDv* name for the EUR/USD currency pair, it is necessary to put *EURUSDv* into the *Map to* field in the Tickstory export settings.

For a proper data acquisition from the Tickstory, the broker's configuration file must be loaded. The configuration file that was used for the data acquisition can be found on the enclosed CD.

4.2 Model Training

Having all the necessary data available the training can be started. MetaTrader 4 offers an option to use a genetic algorithm for the optimization. This is very useful, as it can decrease the amount of required parameter configuration performance measurements from billions down to about 10 000. Another means how to accelerate the processing is by setting an optimization limit. When the limit is reached, the measurement is stopped and the result is thrown away. As all the trainings will be run with an initial balance of 10 000 €, I decided that a suitable limit for stopping of the test should be the balance dropping below 9000 €.

Before every training all the parameters must be set carefully. This includes the model, training start and end, timeframe (period), spread, optimization, initial deposit and its currency, allowed position types, optimized parameter, genetic algorithm, input parameters ranges and optimization limits.

Generic Training Parameters

In this section the parameters that are the same for every algorithm are defined.

- Symbol: EURUSDv
- Model: Every tick
- Spread: 2
- Optimization: True
- Initial deposit and Currency: 10 000 EUR
- Positions: Long & Short
- Optimized parameter: Balance
- Genetic algorithm: True
- Optimization limit: Balance minimum checked and set to 9 000
- For one month training => From: 2015.02.01; To: 2015.02.28
- For one week training => From: 2015.02.21; To: 2015.02.28

Geedo-specific Training Parameters

- Period: H1

The rest of the training parameters for Geedo is shown in Figure 4.2. In order to avoid adjusting them manually, the user can load the configuration file found on the enclosed CD.

Geedo Training Results

The results summarized in Figures 4.3 and 4.4 can also be found in a text file on the enclosed CD.

Genie-specific Training Parameters

- Period: M1

The rest of the training parameters for Genie is shown in Figure 4.5. In order to avoid adjusting them manually a user can load the configuration file found on the enclosed CD.

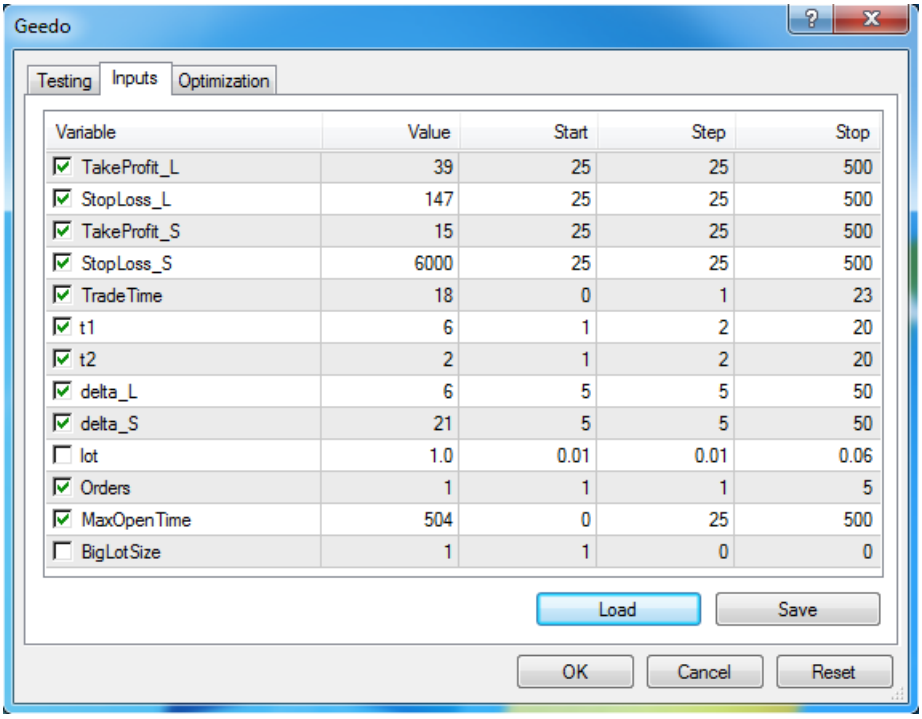

Figure 4.2: Geedo training parameters

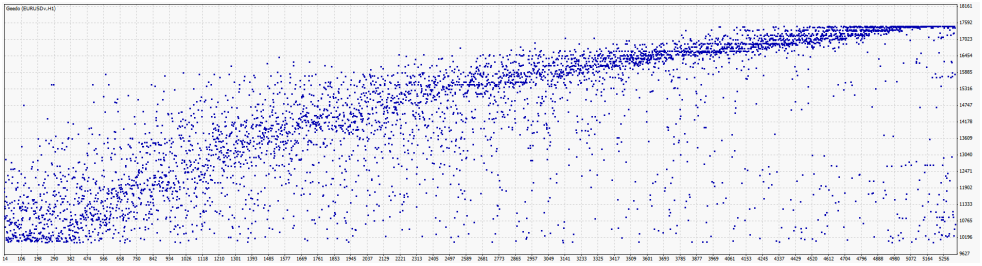

Figure 4.3: Geedo one month training results

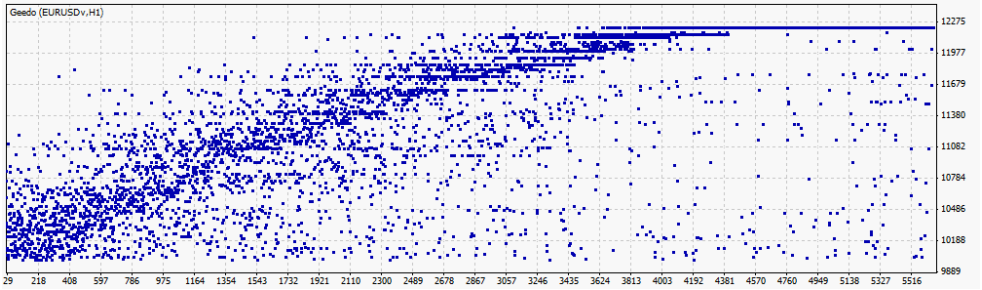

Figure 4.4: Geedo one week training results

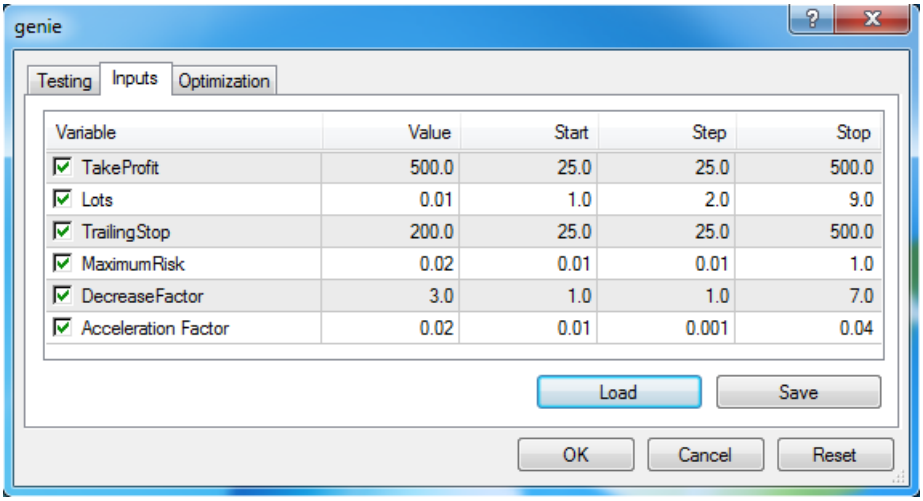

Figure 4.5: Genie training parameters

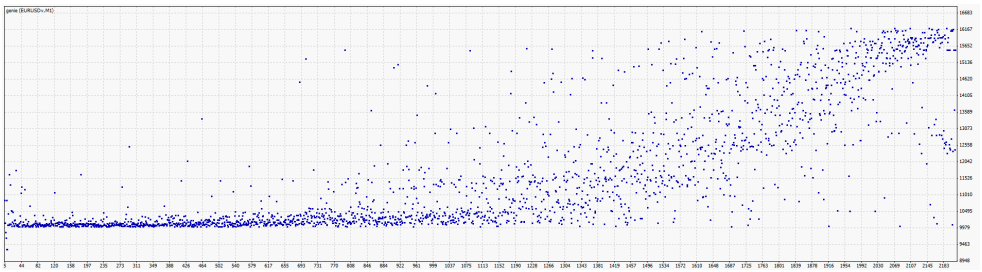

Figure 4.6: Genie one month training results

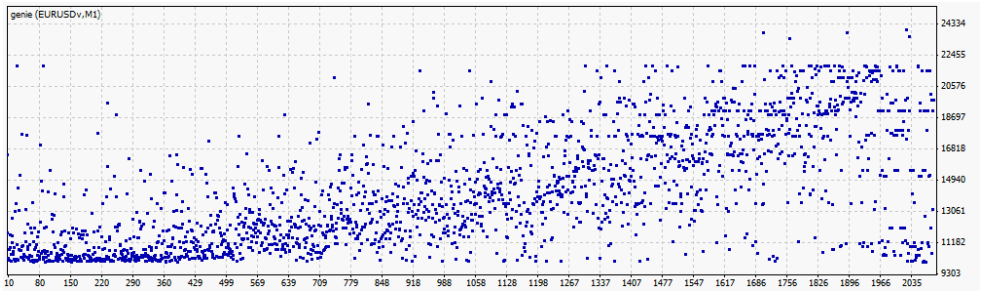

Figure 4.7: Genie one week training results

Genie Training Results

The results summarized in Figures 4.6 and 4.7 can also be found in a text file on the enclosed CD.

My-EA-specific Training Parameters

- Period: M1

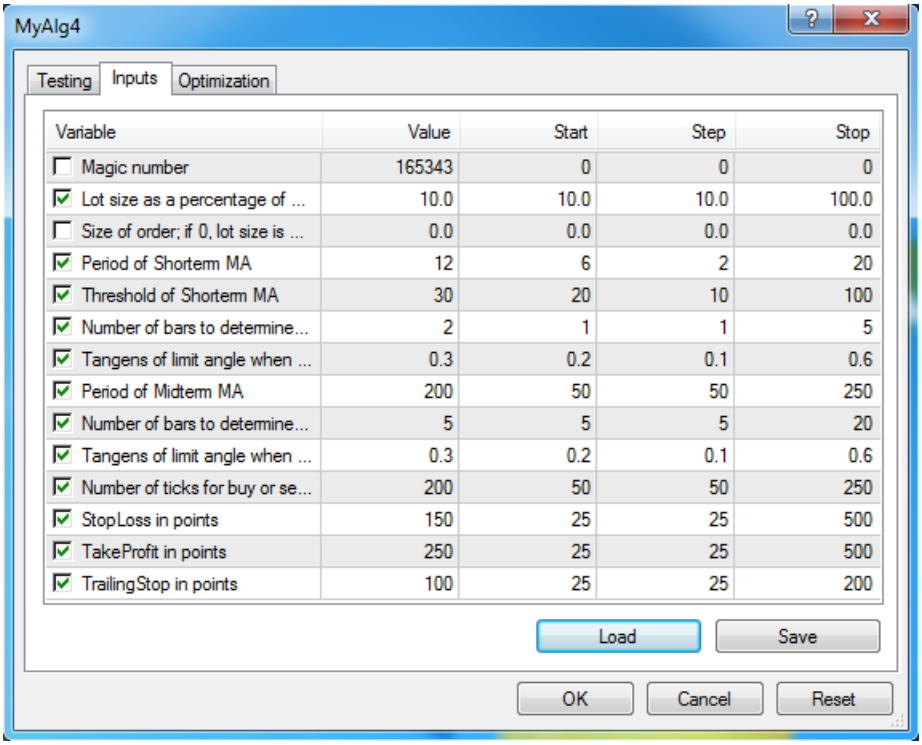

Figure 4.8: My Expert Advisor training parameters

The rest of the training parameters for my Expert Advisor is shown in Figure 4.8. In order to avoid adjusting them manually a user can load the configuration file found on the enclosed CD.

My EA Training Results

The results summarized in Figures 4.9 and 4.10 can also be found in a text file on the enclosed CD.

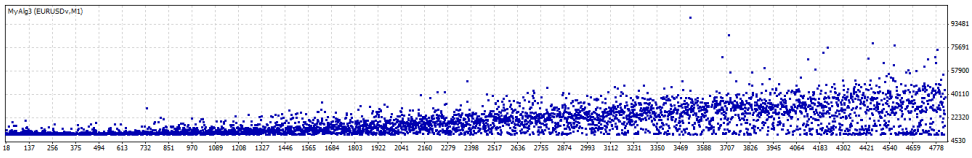

Figure 4.9: My Expert Advisor one month training results

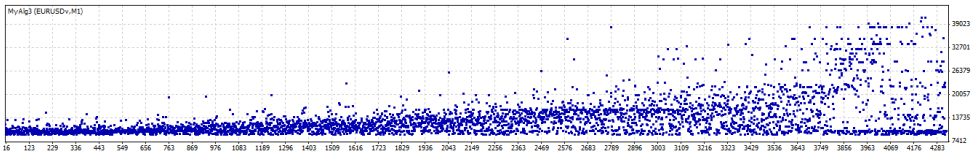

Figure 4.10: My Expert Advisor one week training results

4.3 Model Testing

As mentioned before, the model testing consists of running measurements on five consecutive weeks. Only profitable (and those whose profit is equal to 0) parameter configurations from he training are used for the testing. This is impossible to achieve in the MetaTrader's graphic user interface (GUI). Therefore, all the discussed Expert Advisors were slightly modified for the testing purposes. This modification includes the loading of configuration parameters from a file (not GUI) in the *OnInit()* function and the saving of the results into another file in the *OnTester()* function. The modified EAs for the testing purposes can be found on the enclosed CD.

Generic Testing Parameters

In this section the parameters that are the same for every algorithm are defined.

- Symbol: EURUSDv
- Model: Every tick
- Spread: 2
- Optimization: True
- Initial deposit and Currency: 10 000 EUR
- Positions: Long & Short
- Optimized parameter: Balance
- Genetic algorithm: False
- Optimization limit: None
- For week 1 testing => From: 2015.03.01; To: 2015.03.07
- For week 2 testing => From: 2015.03.08; To: 2015.03.14
- For week 3 testing => From: 2015.03.15; To: 2015.03.21

- For week 4 testing => From: 2015.03.22; To: 2015.03.28
- For week 5 testing => From: 2015.03.29; To: 2015.04.04

Geedo-specific Testing Parameters

- Period: H1

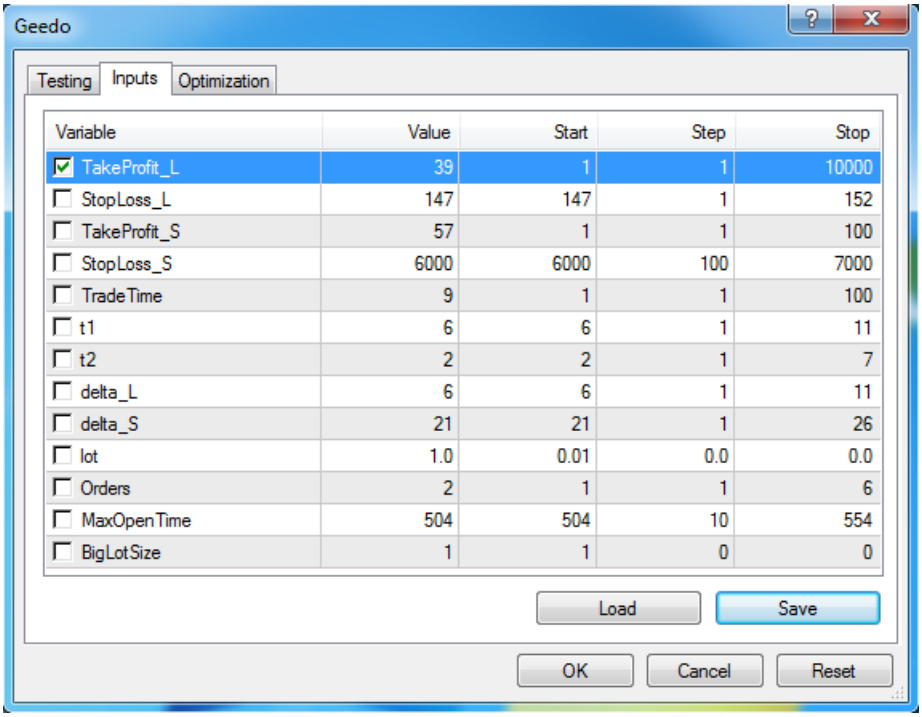

Figure 4.11: Geedo testing parameters

The rest of the testing parameters for Geedo is shown in Figure 4.11. Basically, when it comes to the input parameters for the testing, the only requirement is that the optimization is checked for one parameter only and that this parameter has more steps than the number of tested configurations. This parameter is used as an iterator. Tested configuration will be loaded automatically from a file in the *OnInit()* method. Again, the configuration file can be found on the enclosed CD.

Genie-specific Testing Parameters

- Period: M1

The rest of the testing parameters for Genie is shown in Figure 4.12. The same way as in case of the Geedo testing parameters (see Section 4.3), the

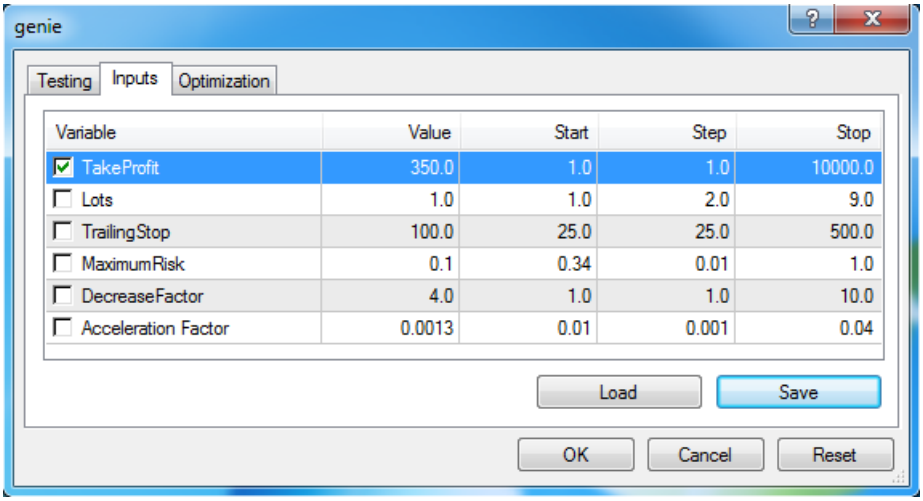

Figure 4.12: Genie testing parameters

optimized parameter is used only as an iterator. Again, the configuration file can be found on the enclosed CD.

My-Alg-specific Testing Parameters

- Period: M1

The rest of the testing parameters for my Expert Advisor is shown in Figure 4.13. The same way as in case of the Geedo testing parameters (see Section 4.3), the optimized parameter is used only as an iterator. Again, the configuration file can be found on the enclosed CD.

4.4 Results

The measured data from the testing are used to compare all three EAs performance-wise.

4.4.1 First Comparison

The first comparison shown in Figure 4.14 shows the performance of each EA in the both training periods for every week. The performance in this figure is determined as the number of profitable configurations out of all the testing configurations (i.e. out of all the profitable configurations from the training).

Figure 4.14 shows the Genie EA not trading profitably neither when trained on the one-week period nor on the one-month period. This means that the model is weak and does not contain enough parameters necessary to set its behavior correctly.

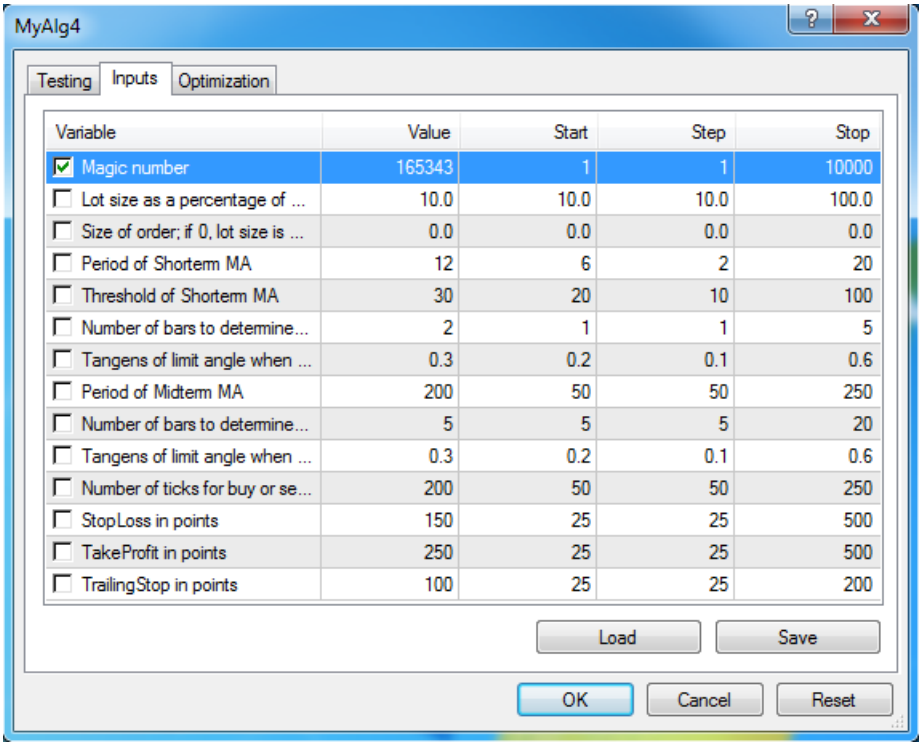

Figure 4.13: My Expert Advisor testing parameters

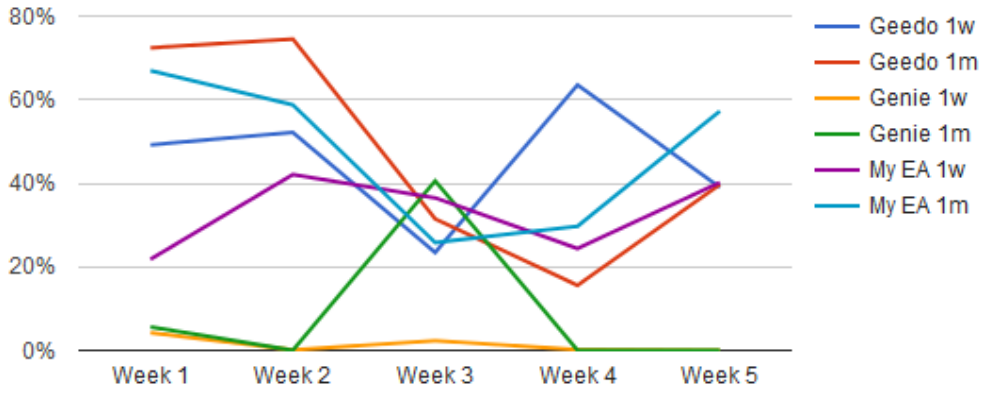

Figure 4.14: Percentage of profitable trades over five weeks testing

The other two EAs delivered plausible results regardless of the training period.

4.4.2 Second Comparison

Then, the set of the top five percent of the most profitable configurations was taken from each EA and a training period combination. Figure 4.15 shows the

percentage of the lossy and the profitable configurations in the testing among the chosen set.

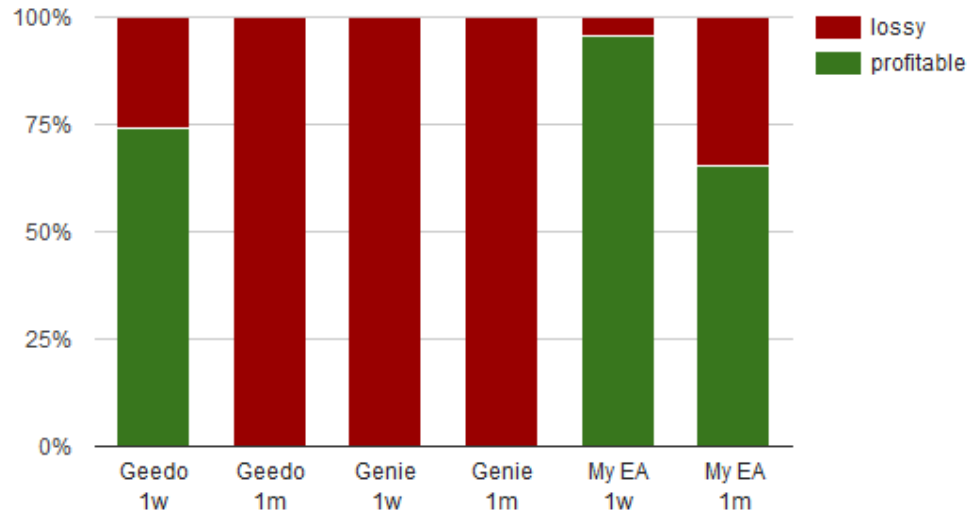

Figure 4.15: Top 5% training configurations testing results

Figure 4.15 shows that not a single one from the top 5% configurations from the training were profitable during the testing period for Geedo trained on the one-month period. This confirms a very important fact about the EA training – even the most profitable EA from the training does not ensure profits during the testing or the live trading. One of the possible reasons for this is, for example, model overlearning. Attempting to determine the configuration quality solely based on the training is a trap that many beginners fall into. The absence of a simple testing functionality in MetaTrader 4 worsens the situation.

As the Genie EA does not trade (see Section 4.4.1), the results of this EA can be considered to be insignificant.

My EA trained on the one-week period has interestingly good results. However, further research would be necessary to find out the exact reason for this.

4.4.3 Best Configurations

Two methods were used to determine the best configurations for each period and algorithm. The first one uses the average of 5 weeks discounted by the $(1 - drawdown)$ factor. The second method works similarly. It uses the weighted average instead of the simple one. The weights of weeks one to five are 1, 0.7, 0.5, 0.3 and 0.2 respectively.

The *drawdown* is an interesting factor as it is an indicator of how much the equity dropped during the course of trading. Generally, the configurations

with a lower drawdown are safer. However, these configurations usually yield lower profits. The goal was to find an exception to the rule – a configuration that has both high profits and a low drawdown.

The best configuration for each EA and each training period can be seen in Tables 4.1 and 4.2.

Table 4.1: Best configurations based on the weighted average discounted by $(1 - drawdown)$. Geedo 1w parameters: TakeProfit_L=500.00, StopLoss_L=425.00, TakeProfit_S=425.00, StopLoss_S=500.00, TradeTime=15.00, t1=7.00, t2=1.00, delta_L=5.00, delta_S=40.00, Orders=5.00, MaxOpenTime=475.00, lot=1.00, BigLotSize=1.00. Geedo 1m parameters: TakeProfit_L=450.00, StopLoss_L=475.00, TakeProfit_S=400.00, StopLoss_S=375.00, TradeTime=16.00, t1=7.00, t2=1.00, delta_L=20.00, delta_S=50.00, Orders=1.00, MaxOpenTime=25.00, lot=1.00, BigLotSize=1.00. Genie 1w parameters: TakeProfit=250.00, Lots=7.00, TrailingStop=475.00, MaximumRisk=0.01, DecreaseFactor=6.00, Step=0.037. Genie 1m parameters: TakeProfit=125.00, Lots=1.00, TrailingStop=500.00, MaximumRisk=0.01, DecreaseFactor=4.00, Step=0.037. My EA 1w parameters: i_stopLoss=275.00, i_takeProfit=425.00, i_trailingStop=200.00, i_relativeLotSize=100.00, i_MAshortTermPeriod=6.00, i_MAshortTermThreshold=40.00, i_MAshortTermTrendBars=1.00, i_MAshortTermTrendToleranceAngleTan=0.50, i_MAmidTermPeriod=150.00, i_MAmidTermTrendBars=10.00, i_MAmidTermTrendToleranceAngleTan=0.60, i_flagTimer=250.00, i_magicNumber=165343.00, i_lots=0.00. My EA 1m parameters: i_stopLoss=225.00, i_takeProfit=425.00, i_trailingStop=200.00, i_relativeLotSize=100.00, i_MAshortTermPeriod=14.00, i_MAshortTermThreshold=90.00, i_MAshortTermTrendBars=5.00, i_MAshortTermTrendToleranceAngleTan=0.40, i_MAmidTermPeriod=200.00, i_MAmidTermTrendBars=20.00, i_MAmidTermTrendToleranceAngleTan=0.60, i_flagTimer=250.00, i_magicNumber=165343.00, i_lots=0.00.

	Train	Week 1	Week 2	Week 3	Week 4	Week 5	Testing profit
Geedo 1w	830.23	855.89	2000.71	934.24	455.72	-675.55	3571.01
Geedo 1m	2079.6	996.61	682.15	1261.26	1254.4	1459.6	5654.02
Genie 1w	49.21	-0.24	-39.45	-4.22	-150.12	-273.29	-467.32
Genie 1m	10.28	1.12	-15.11	-16.07	-111.42	-295.46	-436.94
My EA 1w	119.16	-5640.82	152773.3	-1450.42	-9399.11	-3752.05	132530.9
My EA 1m	24096.54	4062.22	19638.63	828.63	774.25	4369.71	29673.44

Table 4.2: Best configurations based on the average discounted by $(1 - drawdown)$. Geedo 1w parameters: TakeProfit_L=375.00, StopLoss_L=200.00, TakeProfit_S=350.00, StopLoss_S=500.00, TradeTime=17.00, t1=7.00, t2=5.00, delta_L=10.00, delta_S=5.00, Orders=4.00, MaxOpenTime=475.00, lot=1.00, BigLotSize=1.00. Geedo 1m parameters: TakeProfit_L=450.00, StopLoss_L=475.00, TakeProfit_S=400.00, StopLoss_S=375.00, TradeTime=16.00, t1=7.00, t2=1.00, delta_L=20.00, delta_S=50.00, Orders=1.00, MaxOpenTime=25.00, lot=1.00, BigLotSize=1.00. Genie 1w parameters: TakeProfit=250.00, Lots=7.00, TrailingStop=475.00, MaximumRisk=0.01, DecreaseFactor=6.00, Step=0.037. Genie 1m parameters: TakeProfit=75.00, Lots=7.00, TrailingStop=475.00, MaximumRisk=0.01, DecreaseFactor=7.00, Step=0.034. My EA 1w parameters: i_stopLoss=300.00, i_takeProfit=400.00, i_trailingStop=200.00, i_relativeLotSize=100.00, i_MAshortTermPeriod=6.00, i_MAshortTermThreshold=40.00, i_MAshortTermTrendBars=1.00, i_MAshortTermTrendToleranceAngleTan=0.30, i_MAmidTermPeriod=50.00, i_MAmidTermTrendBars=20.00, i_MAmidTermTrendToleranceAngleTan=0.60, i_flagTimer=250.00, i_magicNumber=165343.00, i_lots=0.00. My EA 1m parameters: i_stopLoss=225.00, i_takeProfit=425.00, i_trailingStop=200.00, i_relativeLotSize=100.00, i_MAshortTermPeriod=14.00, i_MAshortTermThreshold=90.00, i_MAshortTermTrendBars=5.00, i_MAshortTermTrendToleranceAngleTan=0.40, i_MAmidTermPeriod=200.00, i_MAmidTermTrendBars=20.00, i_MAmidTermTrendToleranceAngleTan=0.60, i_flagTimer=250.00, i_magicNumber=165343.00, i_lots=0.00.

	Train	Week 1	Week 2	Week 3	Week 4	Week 5	Testing profit
Geedo 1w	1036.49	343.02	1337.87	1169.24	637.58	821.21	4308.92
Geedo 1m	2079.6	996.61	682.15	1261.26	1254.4	1459.6	5654.02
Genie 1w	49.21	-0.24	-39.45	-4.22	-150.12	-273.29	-467.32
Genie 1m	45.44	-69.2	-51.31	37.35	-115.27	-178.29	-376.72
My EA 1w	1368.81	-2713.77	171757	-8086.08	9276.27	6667.03	176900.45
My EA 1m	24096.54	4062.22	19638.63	828.63	774.25	4369.71	29673.44

Conclusion

The purpose of the thesis was to study existing trading algorithms for foreign exchange market trading. Furthermore, an implementation of an own robot was to be delivered. Performance-wise comparison based on historic data was to be realised.

Firstly, the most significant parameters of broker selection were discussed. The reader was advised which parameters he should focus on in order to find the most suitable broker for his needs while avoiding fraudulent brokers. This is the first step to increase probability of profits. Also, the necessity of risk management was stressed by recommending the use of stop orders and hedging.

Secondly, main features of MQL4 language were introduced and explained in detail, as it is necessary to know the details of the language in order to be able to unleash all of its capabilities.

Afterwards, the code of two publicly available expert advisors were analyzed showing varying qualities of free robots. The analysis was beneficial by pointing out the mistakes that should be avoided.

Furthermore, a new EA was implemented with focus on code clarity and reusability, so that it can be utilised as a template for future development.

Next, the two previously analyzed EAs and my new EA were analyzed performance-wise. The analysis was based on model learning and evaluation, which consists of training and testing. The solution of the absence of model testing feature in MetaTrader4 was to tweak training feature of MetaTrader4 in order to be able to load parameter configuration from file instead of GUI. This way, it was made possible to test only parameter configurations which resulted from model training.

Having obtained the testing data, the EAs were compared to each other. A slightly surprising fact is that poorly written (when it comes to coding style) algorithm Geedo significantly outperformed the better written Genie.

Also, the necessity for model testing was confirmed by proving that many top-performing parameter configurations in training were lossy in testing.