MQL5 PROGRAMMIEREN LERNEN – EAS SCHREIBEN EINFACH ERKLÄRT

Hey Insider,
das hier ist einmal wieder ein Selbstexperiment. In unserer Reihe „Metatrader Secrets“ geht es vor allem um die kleinen „Gadgets“, welche in der Software versteckt sind. Da wir im Bereich des Aktienhandels auch automatisierte Tools verwenden, die uns einen Überblick über die Marktlage verschaffen, wollten wir dies auch verstärkt in kleineren Zeiteinheiten in die Währungsmärkte implementieren. Die Programmiersprache R, die wir normalerweise auch für unsere Online-Applikationen verwenden, kann nur umständlich in die MT5 Umgebung eingebunden werden. Einer von uns, sollte daher innerhalb einer Woche, MQL5 erlernen. Welche Quellen wir verwendet haben, um uns Informationen zu der Sprache zu beschaffen, was unsere ersten Testprogramme waren und wie man Expert-Advisors ohne Programmierkenntnisse erstellen kann, wollen wir euch mit dieser Serie näher bringen.

DIE ERSTEN SCHRITTE MIT MQL5 – ÖFFNEN DES MQL5 EDITORS – EA GENERATOR

Der erste Tag.

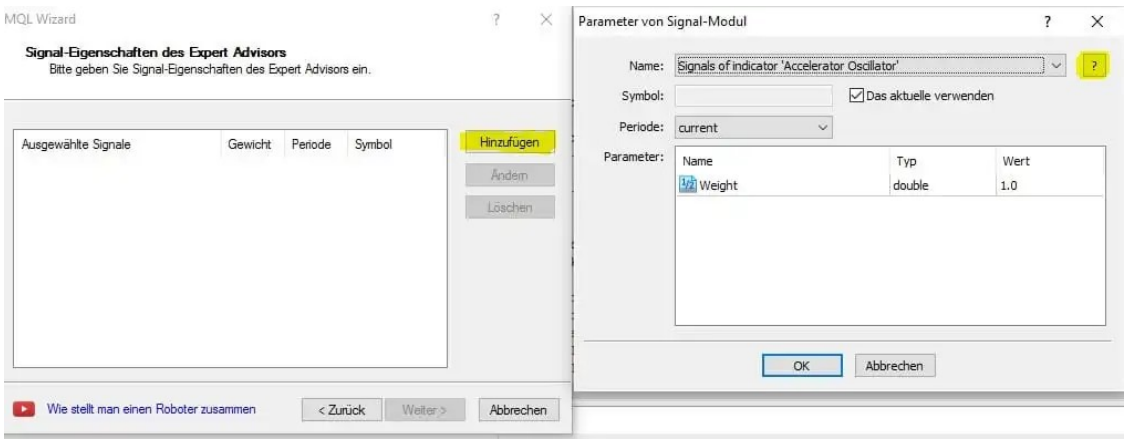
Zu Beginn ist es wichtig, sich einen Überblick über die allgemeine Syntax der Sprache zu verschaffen. Man muss aber auch entscheiden, was man eigentlich mithilfe des EAs erreichen möchte.

Wenn der EA nur einfache, „klassische“ Einstiege handeln soll, so ist es durchaus möglich, ganz ohne eigene Programmierarbeit ans Ziel zu gelangen. Auch wir haben diesen Ansatz versucht und uns dem EA-Generator des Metatrader 5 bedient. In diesem kann man ganz ohne zu Programmieren einfach via Drag and Drop Indikatoren und Signale auswählen, das Risikomanagement sowie die Stop Loss und Take Profit Werte eingeben um sich

den Code automatisch erstellen zu lassen. Um dieses doch recht unbekannte Feature zu nutzen, ist lediglich der MT5-Editor zu starten. Dies kann im Metatrader 5 durch das Klicken auf das folgende Symbol geschehen:

Daraufhin öffnet sich die Oberfläche des Editors. Um nun zum EA-Generator zu gelangen, der den gewünschten EA automatisch erstellt, ist lediglich bei Daten->Neu->Expert Advisor (generieren)->Weiter zu klicken. Daraufhin wird man Schritt für Schritt durch den Ablauf geleitet. Der Generator bietet einem vorgefertigte Ein- und Ausstiegssignale, sowie prozentuale Risikoeinstellungen und Trailling-Stop Optionen. Mithilfe der vorinstallierten Auswahlmöglichkeiten lassen sich bereits einige Hundert EAs erstellen. Weitere lassen sich über den MQL5 Shop erwerben.

Zu jedem der von Vornherein verfügbaren Signale ist eine ausführliche Dokumentation vorhanden. Diese lässt sich über das Fragezeichen bei der Auswahl der Indikatoren öffnen. Zur Auswahl der Indikatoren gelangt man durch einen Klick auf den „Hinzufügen“ Button.



Als nächsten Schritt kann man Trailing-Stop Optionen zu dem Expert-Advisor

hinzufügen. Dabei lassen sich verschiedene Trailling-Stop Methoden heranziehen – von festen Abständen bis hin zu einem Trailing-Stop hinter einer SMA ist alles möglich. Um den Expert-Advisor abzuschließen, werden am Ende des Generators noch die Risikoparameter abgefragt. Im Speziellen sind dies die Positionsgrößenschritte und das prozentuale Risiko. Sobald man nun auf „Fertigstellen“ klickt, wird der EA einsatzbereit ausgegeben und dieser kann direkt getestet werden. Dazu kann man beispielsweise den Strategietester des Metatrader 5 nutzen, (diesen öffnet man mithilfe der Tastenkombination STRG+R in Metatrader) welcher den erstellten EA auf vergangenen Kursverläufen testet. Dies kann sowohl visuell als auch nicht visuell erfolgen. Beim visuellen Test kann man dem Experten direkt bei der Arbeit zuschauen und sieht so auch, ob die ursprünglich gewollte Signallogik auch wirklich umgesetzt wird oder ob das Programm kleinere Fehler aufweist. Mehr zum Thema „Testen einer Strategie“ werden wir in einem Video betrachten. Tragt euch am besten in unseren Newsletter ein, um unsere besten Veröffentlichungen nicht mehr zu verpassen, sowie monatlich marktrelevante Informationen zu erhalten.

Um allerdings selbstständig programmieren zu können, ist es notwendig sich einen ersten Eindruck von der Syntax der Sprache zu machen und sich ein Handbuch zu organisieren, in dem die wichtigsten Befehle dokumentiert sind. Das Handbuch war schnell gefunden und wird in deutscher und englischer Sprache von der Firma Metaquotes zur Verfügung gestelllt: MetaTrader 5 Handbuch (https://www.mql5.com/files/pdf/mql5_german.pdf). Dieses stolze 4369 Seiten umfassende Nachschlagewerk enthält alle wichtigen Befehle der Programmiersprache und liefert dazu noch Beispiele und Dokumentationen. Ein solches Grundlagenwerkes stellte sicher, dass wir die richtigen Befehle sowie deren Verwendung parat hatten und nachvollziehen konnten. Weiterhin hielten wir es für sinnvoll, uns noch einen sehr gut dokumentierten Code zu suchen. In Verbindung mit dem Nachschlagewerk konnten wir so nachvollziehen, wie die Programmiersprache funktioniert und machten uns mit der Syntax der If-Statements und Schleifen vertraut. Wir nutzen hierfür Außerdem interessierten wir uns noch für die Funktionsweise der Orderaufgabe im Detail (hier geht es zur Dokumentation: Ein Paar Testläufen eines EAs aus dem Internet reichten aus, um uns mit dem EA-Strategie Tester vertraut zu machen. Für heute hatten wir uns den Feierabend mehr als verdient.

MQL5 SYNTAX – PARALLELEN ZU C & C++

Der zweite und dritte Tag.

Nachdem der erste Tag viel Neues mit sich brachte, war klar, dass das tiefere Eindringen in die Syntax der Sprache essentiell ist. Da MQL5 an die Programmiersprache C++ angelehnt ist und diese eine wesentlich bessere Dokumentation als MQL5 hat, setzen wir uns ein oder zwei Tage mit dieser Sprache und ihren Eigenarten auseinander. Auf der Plattform von Sololearn gibt es einen kostenfreien Kurs zu C++. (Dieser kann hier abgerufen werden: http://www.sololearn.com/Play/CPlusPlus). Der Kurs ist sowohl als App als auch als Online-Version verfügbar und synchronisiert sich automatisch, sodass der aktuelle Fortschritt immer und überall abrufbar ist. Die in Browser Version hat man aus Erfahrung schneller bearbeitet und ist während der Bearbeitung aufmerksamer als bei der Nutzung der App für unterwegs, aber es geht ja auch nur um ein Gefühl für die Sprache. Wie schreibt man Methoden und wendet diese an? Wie werden Variablen deklariert? Wie funktioniert ein if-Statement? Wie wende ich eine Funktion auf ein Objekt an? Wie lade ich Klassen aus der Standardbibliothek?

All diese Fragen sind typisch und mehr als nur berechtigt. Das Problem ist, dass es zu MQL5 dazu eben keine so intuitiven Antworten und Lernmöglichkeiten gibt.

UPDATE: Vor kurzem sind wir auf den folgenden MQL5 Kurs gestoßen. Da Programmieren am anfang etwas ernüchternd sein kann, sind wir uns sicher, dass es euch hilft, in einen aktiven Kurs eingebunden zu sein, der auch E-Mail Support anbietet, was in manchen Debugging-Fragen sicherlich hilfreich sein wird.

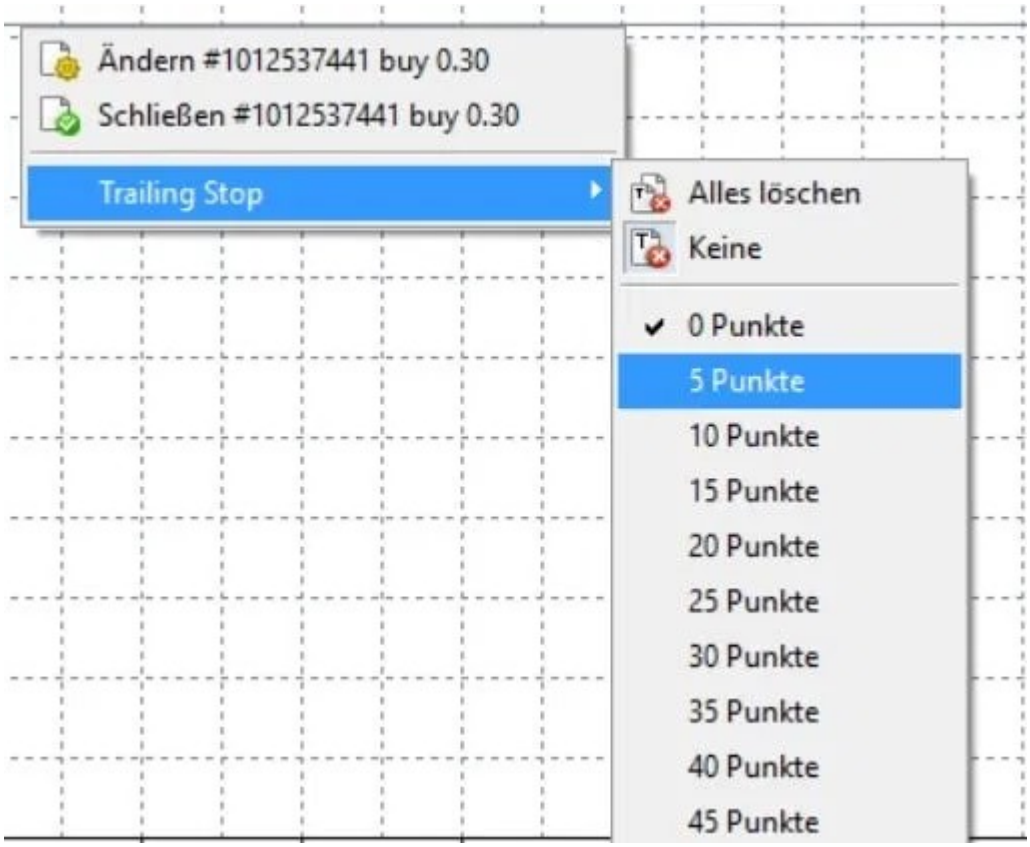
Zum Golden Goose MQL Kurs

UNSER ERSTER EA – MQL5 TRAILING STOP

Tag vier und fünf

Es war nun an der Zeit, das Gelernte anzuwenden und den ersten eigenen kleinen EA zu schreiben. Doch was für ein Projekt ist als Einstieg adäquat? Da

uns zu dieser Zeit das Thema Positionsmanagement sehr interessierte, entschieden wir uns für das Programmieren eines Trailling Stop EAs. Natürlich ist eine Trailing Stop Funktion bereits im Metatrader 5 vorinstalliert und lässt sich über einen Rechtsklick auf die offene Position einstellen:



Uns reichten jedoch die Standardeinstellungen nicht aus. Wir wollten auch Indikatoren als Trailling Stop verwenden können und Backtests durchführen, um herauszufinden welcher Stop im gewählten Zeitraum der Optimale war. Also starteten wir das „einfache Projekt“.

Ziemlich schnell wurde klar, dass das Aufgeben und Modifizieren von Orders und Positionen weitaus komplizierter ist als erwartet. Der normale Weg durch die Metratrader Ordermaske, wie er auch in der Sprache MQL5 gegangen werden kann / muss, ist sehr fehleranfällig, kompliziert und unübersichtlich. Daher schauten wir uns um und wurden in den Standard Bibliotheken von MQL5 fündig.

```
#include <Trade\Trade.mqh>
#include <Trade\PositionInfo.mqh>
#include <Trade\SymbolInfo.mqh>
```

Diese Libraries enthalten Funktionen und Methoden, die die Programmierung und die Projektentwicklung enorm vereinfachen. Statt nun alles aufwändig von Hand zu schreiben und zu befüllen, bieten sie einfache und intuitive Zugriffe auf Positionen, Orders und Trade-Parameter.

Nähere Infos zu diesen Libraries fanden wir hier (geordnet nach Erwähnung im Artikel):

https://www.mql5.com/de/articles/138

https://www.mql5.com/de/docs/standardlibrary/tradeclasses/ctrade

https://www.mql5.com/en/docs/standardlibrary/tradeclasses/cpositioninfo

https://www.mql5.com/en/docs/standardlibrary/tradeclasses/csymbolinfo

https://www.mql5.com/en/docs/constants/tradingconstants/positionproperties

Bevor wir nun im Detail in den Code starteten, schauten wir uns den Aufbau und den Ablauf eines solchen Programmes an.

Zu Beginn im Kopf des EAs werden Informationen über die Rechte und die Lizenz des EAs gemacht. Gleichfalls wird angegeben, wer den EA programmiert hat. Neben diesen Informationen wird der Output hinzugefügt, der später beim Start des EAs im Metatrader 5 im Description Teil zu lesen ist.

Direkt im Anschluss werden die Klassen aus der Standardbibliothek geladen, die die Funktionen und Methoden enthalten, die wir zur Entwicklung von einfachen und eleganten Code brauchen

Im Anschluss werden die Variablen und Inputs deklariert, die zum Betreiben des Programms benötigt werden. Inputs sind Variablen, die der Nutzer des EAs beim Starten im MT5 einstellen kann. Alle Variablen, die hier erstellt werden, sind für den kompletten Code, also das ganze Programm verfügbar. Alle folgenden Funktionen können darauf Zugreifen und mit ihnen arbeiten. In einem extra Video werden wir sehen, wie man durch die Inputs, einen EA optimieren kann. Auf diese Art und Weise lassen sich Informationen über das Handelssystem sammeln und man kann durch stabile, optimierte Parameter die Performance des Handelssystems steigern.

Nachdem alle Variablen bekannt gemacht wurden, folgte die Funktion OnInit(). In dieser, werden alle Indikatoren und Einstellungen geladen, die beim Start des EAs benötigt werden. So werden hier zum Beispiel Arrays, also Daten Matrizen, zu Zeitreihen modifiziert. Dies ist für die spätere Signalerzeugung notwendig. Bei einem Cross Over eines Moving Averages – also dem Schnitt von gleitenden Durchschnitten– muss der schnelle Durchschnitt in der letzten Periode zum Beispiel unterhalb des langsamen MAs liegen und in der aktuellen Periode entsprechend oberhalb. Damit diese Zugriffe möglich sind, bedarf es der Zeitreihenform der Indikatoren. Außerdem werden die , also die Variablen, die die Daten der Indikatoren übernehmen, in dieser Funktion initiiert und befüllt.

```
9 //+------------------------------------------------------------------+
```

```
10 //| Expert initialization function                                   |
11 //+------------------------------------------------------------------+
12 int OnInit()
13   {
14 //---
15
16 //---
17    return(INIT_SUCCEEDED);
18   }
```

Nachdem die wichtigsten Einstellungen in der Startfunktion OnInit() erfolgten, muss nun festgelegt werden, was beim Entfernen der EA vom Chart (entspricht dem Abschalten) passiert. Um diese Schritte festzulegen, verwendet man die OnDenit() Funktion. In dieser Funktion werden erzeugte Arrays und Zeitreihen zur Löschung freigegeben, sowie Indikatoren entfernt und Einstellungen zurückgesetzt. Dies ist notwendig, um den vom EA belegten Speicher wieder sauber freizugeben und die Chart sowie das Handling des MT5 wieder in den Ursprungszustand zurück zu bewegen.

```
19 //+------------------------------------------------------------------+
20 //| Expert deinitialization function                                 |
21 //+------------------------------------------------------------------+
22 void OnDeinit(const int reason)
23   {
24 //---
25
26   }
```

Der Rahmen für das Programm war erstellt. Wir beschäftigten uns nun mit den beiden Funktionen, welche für die Signal-Generierung sowie den Umgang mit offenen Positionen interessant werden. Die beiden Funktionen sind OnTick() und OnTrade(). OnTick() wird immer ausgeführt, sobald sich eine Veränderung im Symbol, welches der EA ausliest, ereignet. Die OnTrade()- Funktion ähnelt der OnTick() Funktion. Dieser Teil des Codes wird nur ausgelesen, sofern eine Position im Symbol offen ist.

```
27 //+------------------------------------------------------------------+
28 //| Expert tick function                                             |
29 //+------------------------------------------------------------------+
30 void OnTick()
31   {
32 //---
33
34   }
35 //+------------------------------------------------------------------+
36 //| Trade function                                                   |
37 //+------------------------------------------------------------------+
38 void OnTrade()
39   {
40 //---
41
42   }
```

Da der Standardaufbau eines EAs nun bekannt ist, wollen wir die einzelnen Funktionen befüllen und der Maschine „Leben“ einhauchen.

Unser Ziel ist es, einen Trailling Stop EA zu entwickeln, der nach dem Eröffnen

einer Position automatisch einen Stop setzt und diesen in Folge kontinuierlich nachzieht. Das Nachziehen soll dabei mit einem konstanten Abstand sowie immer beim Erreichen eines neuen Hochs geschehen. Wir brauchen also:

- Einen Test, ob eine Position eröffnet wurde und was ihre Parameter sind
- Einen Weg, den aktuellen Stop anhand des momentanen Marktkurses zu errechnen
- Eine Möglichkeit zur Bestimmung der Nachkommstellen, die der Markt auf den EA anwenden wird, um den eingestellten Stop entsprechend anzuwenden
- Eine Überprüfung, ob der neue Stop höher bzw. niedriger ist als der aktuelle (abhängig davon ob es sich um eine Long oder Short Position handelt

Zunächst werden dafür alle Variablen und Klassen hinzugefügt und erstellt:

```
#include <Trade\Trade.mqh>
#include <Trade\PositionInfo.mqh>
#include <Trade\SymbolInfo.mqh>

input double StopDist;

double NewStop;
double CurStop;

CTrade m_Trade;
CPositionInfo myposition;
```

Wofür genau double, int, sowie Klammer- und Punktsetzung etc. stehen, soll nicht Teil dieses Artikel sein und kann im entsprechenden C++ Tutorial (als alternative ein schönes Buch) schnell erlernt und nachgeschlagen werden.

Im Code Ausschnittieht man, dass zwei double Variablen global deklariert werden: NewStop, CurStop. NewStop wird den aktuellen rechnerischen Stop enthalten und CurStop wird den aktuellen Stop der offenen Position zwischenspeichern. Wir benötigen beide Variablen, um später vergleichen zu können, welcher Stop das geringere Risiko in sich trägt und entsprechend den von uns gewünschten Stop als Stop für die Position zu wählen. Die Variablen m_Trade und myposition sind Objekte, welche mithilfe der Standardbibliotheken erzeugt wurden. Das Objekt m_Trade spricht den aktuellen Trade an, der im Symbol geöffnet ist. myposition enthält Informationen über die aktuelle Position, wie beispielsweise den aktuellen Stop Loss, Take Profit, das Volumen etc.

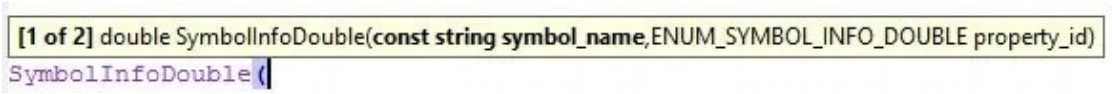
Da wir für einen Trailing Stop, keine speziellen Einstellungen vornehmen müssen und aktuell auch keine Indikatoren gewollt sind, können die Funktionen OnInit() sowie OnDenit() ungefüllt gelassen werden. Für unsere Zwecke werden die Funktionen OnTrade() sowie OnTick() besonders interessant. Im Falle eines Trade Szenarios, sowie bei jedem im Symbol entstehenden Tick, wollen wir prüfen, ob unser Trade noch unseren Vorstellungen entspricht und/oder ob wir etwas anpassen müssen. In der Funktion OnTrade() werden wir den initialen Stop der Position festlegen. Diesen benötigen wir, um später unseren Stop trailen zu können, sofern er vorteilhaft im Verhältnis zum aktuellen Stop ist.

Betrachten wir zunächst einmal die Eingaben, welche wir in der OnTrade() Funktion tätigen. Hier soll der initiale Stop gesetzt werden, welchen wir als Referenzwert für unseren Trailling Stop benötigen.

```
//+------------------------------------------------------------------+
void OnTrade()
  {
   double PriceB=SymbolInfoDouble(_Symbol,SYMBOL_BID);
   double PriceS=SymbolInfoDouble(_Symbol,SYMBOL_ASK);
   double Inital_Safe_Stop = StopDist*3;

   double TP=0;

   int Commas=SymbolInfoInteger(_Symbol,SYMBOL_DIGITS);
   double Factor=1/MathPow(10,Commas-1);

   double SL_B=PriceB-(Inital_Safe_Stop*Factor);
   double SL_S=PriceS+(Inital_Safe_Stop*Factor);


   if(PositionSelect(_Symbol))
     {
      if(myposition.PositionType()==POSITION_TYPE_BUY)
        {
         if(myposition.StopLoss()==0)
           {
            m_Trade.PositionModify(_Symbol,SL_B,TP);
           }
        }

      // Same now For Sell Positions
      if(myposition.PositionType()==POSITION_TYPE_SELL)
        {
         if(myposition.StopLoss()==0)
           {
            m_Trade.PositionModify(_Symbol,SL_S,TP);
           }
        }
     }
```

```
   }
//+------------------------------------------------------------------+
```

Der Code ist in diesem Fall nicht auf maximale Effizienz sondern auf Verständlichkeit optimiert. Grundsätzlich sollte zunächst darauf geachtet werden, die Programme zum Laufen zu bekommen und sich dann mit Optimierung und performanter Programmierung zu beschäftigen. Dank leistungsfähiger Hardware kann uns heutzutage auch die ein oder andere unschöne Darstellung verziehen werden. Gehen wir den Code einmal Schritt für Schritt durch. Zunächst wird die OnTrade() Funktion geöffnet. Dann wird der aktuelle Bid und Ask Preis des Marktes in einer Variablen gespeichert. Für diese Abfrage nutzen wir die SymbolInfoDouble() Funktion. In der Funktion selbst nutzen wir _Symbol, um den Markt auszuwählen, auf dem der EA aktuell aktiv ist. Danach geben wir an, welche Information wir vom ausgewählten Symbol abfragen wollen. In unserem Fall eben den Bid und den Ask Preis. Die Funktion gibt, wie der Name es bereits sagt, die Informationen als Double, also als Fließkommazahl aus.



Um die Programmierung zu vereinfachen und zu beschleunigen gibt der Metatrader Editor immer an, welche Informationen der Funktion zu übergeben sind, damit sie problemlos funktioniert. Sollte man wider Erwarten damit nicht weiterkommen, so ist das Googlen der gefragten Funktion meistens zielführend.

Nach der Abfrage der aktuellen Preise des Marktes wollen wir eine Variable für den Initialstop, welcher für nur einen Tick, also bis zur nächsten Kursstellung im Basiswert, im Markt ist und dann sofort durch den Ricchtigen Stop ersetzt wird erstellen. Dies ist nicht zwangsläufig notwendig aber ist ein zusätzlicher Schritt, in dem wir mit Rechenoperationen in Berührung kommen. Dazu multiplizieren wir den Trailling Stop Abstand mit 3 und speichern diese Pip Anzahl ebenfalls als Variable des Typ Double ab. Damit der Stop Loss für jeden Basiswert funktional ist, beziehen wir die Anzahl der Stellen ein, den ein Basiswert hat, denn dies ist nicht für alle Basiswerte gleich. So haben die Yen Paare weniger Stellen als zum Beispiel der EUR/USD. Damit dies keine Probleme bereitet, passen wir unsere Stop Inputs an. Dies passiert durch die folgende kurze mathematische Formel unter Zuhilfenahme der MathPow() Funktion, welche das Potenzieren in MQL5 erlaubt.

Um den initialen Stop Loss für eine Kauf- und eine Verkaufsposition zu bestimmen, ziehen wir im Falle der Kaufposition den initialen angepassten Stop vom Bid-Preis ab, bzw. addieren ihn im Fall einer Verkaufsposition auf den Ask

Preis. Darauf folgt lediglich die Abfrage, ob eine Position im aktiven Markt offen Position eine Kauf oder Verkaufsposition ist. Außerdem wird abgefragt, ob der Stop der Position noch nicht gesetzt ist. Wenn dies der Fall sein sollte, erfolgt eine Anpassung der Order und der initiale Stop Loss wird eingestellt.

Dabei sind folgende Funktionen besonders hervorzuheben:

```
if(myposition.PositionType()==POSITION_TYPE_BUY)
```

Myposition ist ein Objekt, welches durch die Standardklasse SymbolInfo.mqh und CPositionInfo erstellt wurde. Dadurch lassen sich Informationen über offene Positionen leicht durch Methoden wie PositionType() abfragen oder vergleichen.

```
if(myposition.StopLoss()==0)
```

Im nächsten Schritt wurde mit dem selben Objekt aber nun mit der methodischen Abfrage StopLoss() der aktuelle Stop Loss Wert der Position ausgelesen und logisch getestet.

```
m_Trade.PositionModify(_Symbol,SL_S,TP);
```

Zu guter Letzt wird durch das mit Hilfe von CTrade erstellte Objekt und der Methode PositionModify() die offene Position verändert und der Initial Stop greift.

der Stop aber noch nicht dynamisch, sondern statisch auf dem dreifachen Level des Stops, welchen wir als Trailing Stop haben wollen. Das Dreifache ist hier nur Beispielhaft gewählt. Es soll erkennbar seinist, dass der Stop versetzt wird. Auch wenn man dies nur so lange braucht, bis im Markt der nächste Preis festgestellt wurde, ist es ein kleines Erfolgserlebnis diesen Schritt im Chart zu sehen. Es bedeutet, dass der Stop funktioniert.

Damit der Stop sich kontinuierlich anpasst, sofern der Markt in unsere Richtung läuft und somit Gewinne abgesichert werden, bzw. das Risiko reduziert wird, brauchen wir eine Abfrage, welche Tick für Tick prüft, ob der Stop angepasst werden muss und dies bei Bedarf umsetzt. Hier kommt unsere OnTick() Funktion ins Spiel.

```
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
//---
   double PriceB=SymbolInfoDouble(_Symbol,SYMBOL_BID);
   double PriceS=SymbolInfoDouble(_Symbol,SYMBOL_ASK);
```

```
   double TP=0;

   int Commas=SymbolInfoInteger(_Symbol,SYMBOL_DIGITS);
   double Factor=1/MathPow(10,Commas-1);

   double SL_B=PriceB-(StopDist*Factor);
   double SL_S=PriceS+(StopDist*Factor);

   if(PositionSelect(_Symbol))
     {
      if(myposition.PositionType()==POSITION_TYPE_BUY)
        {
         if(myposition.StopLoss()<SL_B)
           {
            m_Trade.PositionModify(_Symbol,SL_B,TP);
           }
        }
      // Same now For Sell Positions
      if(myposition.PositionType()==POSITION_TYPE_SELL)
        {
         if(myposition.StopLoss()>SL_S)
           {
            m_Trade.PositionModify(_Symbol,SL_S,TP);
           }
        }
     }
  }
//+------------------------------------------------------------------
```

Auf den ersten Blick lässt sich sehr schnell erkennen, warum wir erwähnten, dass man dieses Programm durchaus auch hätte effizienter schreiben können. Doch um den Lerneffekt zu vergrößern, sollte die Lesbarkeit und Verständlichkeit an vorderster Stelle stehen.

In der folgenden Kaskade von if() Abfragen wird genau wie bei der Setzung des initialen Stops zuerst geprüft, ob eine Order vorliegt und was für eine Position aktuell läuft. Jedoch erfolgt dann ein Vergleich, ob der aktuelle Stop größer oder kleiner als der Trailing Stop ist. Ist der Trailing Stop vorteilhaft für uns, also das Stop Level bei einer Kaufposition größer beziehungsweise bei einer Short Position kleiner als der aktuelle Stop, so wird die Order mit der PositionModify() Methode verändert, der Stop wird also nachgezogen.

Durch die Nutzung von Indikatoren können noch komplexere Trailing Stop EAs erzeugt werden. Wie Indikatoren in einem EA angewandt werden, wird an den letzten beiden Tagen besprochen.

```
#include <Trade\Trade.mqh>
```

```
#include <Trade\PositionInfo.mqh>
#include <Expert\ExpertTrailing.mqh>
#include <Trade\SymbolInfo.mqh>

input double StopDist;

double NewStop;
double CurStop;

CTrade m_Trade;
CPositionInfo myposition;
//+------------------------------------------------------------------+
//| Expert initialization function
//+------------------------------------------------------------------+
int OnInit()
  {
//---

//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//---

  }
//+------------------------------------------------------------------+
//| Expert tick function
//+------------------------------------------------------------------+
void OnTick()
  {
//---
   double PriceB=SymbolInfoDouble(_Symbol,SYMBOL_BID);
   double PriceS=SymbolInfoDouble(_Symbol,SYMBOL_ASK);

   double TP=0;

   int Commas=SymbolInfoInteger(_Symbol,SYMBOL_DIGITS);
   double Factor=1/MathPow(10,Commas-1);

   double SL_B=PriceB-(StopDist*Factor);
   double SL_S=PriceS+(StopDist*Factor);

   if(PositionSelect(_Symbol))
     {
      if(myposition.PositionType()==POSITION_TYPE_BUY)
        {
         if(myposition.StopLoss()<SL_B)
           {
            m_Trade.PositionModify(_Symbol,SL_B,TP);
           }
```

```
         }
       // Same now For Sell Positions
       if(myposition.PositionType()==POSITION_TYPE_SELL)
         {
          if(myposition.StopLoss()>SL_S)
            {
             m_Trade.PositionModify(_Symbol,SL_S,TP);
            }
         }
      }
   }
//+------------------------------------------------------------------+
void OnTrade()
   {
    double PriceB=SymbolInfoDouble(_Symbol,SYMBOL_BID);
    double PriceS=SymbolInfoDouble(_Symbol,SYMBOL_ASK);
    double Inital_Safe_Stop = StopDist*3;

    double TP=0;

    int Commas=SymbolInfoInteger(_Symbol,SYMBOL_DIGITS);
    double Factor=1/MathPow(10,Commas-1);

    double SL_B=PriceB-(Inital_Safe_Stop*Factor);
    double SL_S=PriceS+(Inital_Safe_Stop*Factor);


    if(PositionSelect(_Symbol))
      {
       if(myposition.PositionType()==POSITION_TYPE_BUY)
         {
          if(myposition.StopLoss()==0)
            {
             m_Trade.PositionModify(_Symbol,SL_B,TP);
            }
         }

       // Same now For Sell Positions
       if(myposition.PositionType()==POSITION_TYPE_SELL)
         {
          if(myposition.StopLoss()==0)
            {
             m_Trade.PositionModify(_Symbol,SL_S,TP);
            }
         }
      }

   }
//+------------------------------------------------------------------+
```

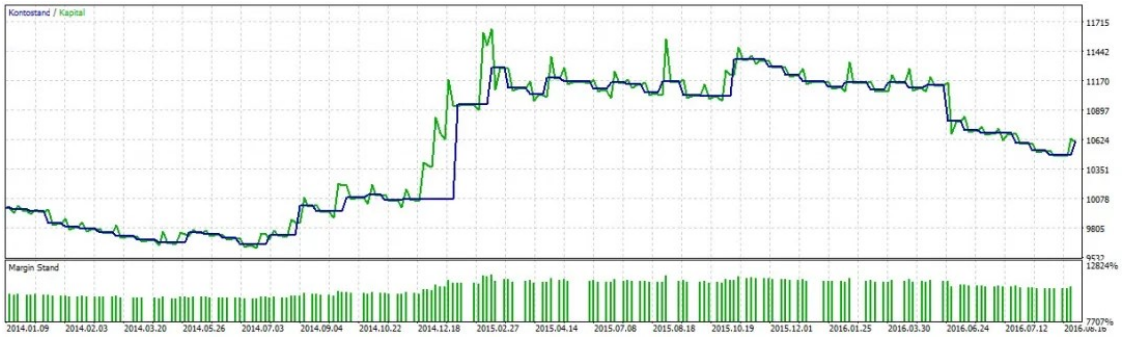
INDIKATOREN IN MQL5 – TRADING

SIGNALE ERSTELLEN

Tag sechs und sieben – Endlich MQL5 Basics beherrschen.

Wir haben gelernt, Positionen zu modifizieren und können Stop Loss und auch Take Profits anpassen (dies funktioniert entsprechend der Stop Loss Anpassung). Es ist an der Zeit Signale zu erzeugen und Trades zu generieren, denn MQL5 ist neben der Anpassung von Positionen und einem damit einhergehenden vereinfachtem Risikomanagement hauptsächlich für das Schreiben von automatisierten Handelssystemen geschrieben worden.

Das folgende Handelssystem ist sehr einfach gebaut. Seine Performance ist auch sicher nicht die beste. Jedoch lässt sich an diesem der Aufbau eines Handelssystems gut erkennen. Die zugrundeliegende Theorie ist der CCI, ein oszillierender Indikator. Wir wollen das Momentum des Marktes handeln und mit dem Trend gehen. Wir gehen im Markt Long sobald der CCI einen Wert von über 100 erreicht und wir shorten sobald der CCI unter -100 fällt. Um unsere Risiken zu limitieren, verlassen wir die Long Position bei CCI Werten unter 95 und die Short Position bei Werten über -95. Damit erzeugen wir eine Asymmetrie, die sich auch in der Performance widerspiegelt. Ein solches Handelssystem sollte im Speziellen in trendstarken Märkten wie dem EUR/USD ab Mitte 2014 bis zur Jahresmitte 2015 gut funktionieren. In einem Backtest ergab sich die folgende Equity-Kurve:



Es lässt sich gut erkennen, dass die Gewinne in Schüben kommen und wir viele kleine Verluste erleiden. Dies entspricht den Erwartungen, welche wir mit dem System verknüpft haben. In dem Zeitraum erzielte der EA insgesamt eine Performance von 6.24%, was nicht besonders überragend ist, aber immerhin macht er Profite.

Es zeigt sich jedoch auch sehr deutlich, dass wir durch den Oszillator immer nur sehr kurz im Markt investiert sind. Die Folge sind hohe Transaktionskosten, sowie nur kurze Bewegungen, die eingefangen werden. Ein solches Problem hätte man bei einem Trendfolge-System nicht. Jedoch sind dabei auch die Risiken beim Eröffnen der Position größer. Das Risikomanagement sowie die Stop Setzung muss komplexer werden, als es bei diesem System der Fall ist.

Hier nun der Quellcode des EAs:

```
//+------------------------------------------------------------
//|                                                       Insi
//|                     Copyright 2016, MetaQuotes Soft
//|                                                 https://ww
//+------------------------------------------------------------
#property copyright "Copyright 2016, MetaQuotes Software C
#property link      "https://www.mql5.com"
#property version   "1.00"

input int CCI_MA;
input int CCI_Upper;
input int CCI_Lower;
input int CCI_Upper_Exit;
input int CCI_Lower_Exit;
input int TrendFilter;

#include <Trade\Trade.mqh>
#include <Trade\PositionInfo.mqh>
#include <Expert\ExpertTrailing.mqh>
#include <Trade\SymbolInfo.mqh>

int CCI_Handle;
int MA_Handle;

double CCI_Val[];
double MA_Val[];

CTrade my_trade;
CPositionInfo my_position;
//+------------------------------------------------------------
//| Expert initialization function
//+------------------------------------------------------------
int OnInit()
  {
//---
   CCI_Handle=iCCI(_Symbol,0,CCI_MA,PRICE_CLOSE);
   MA_Handle = iMA(_Symbol,0,TrendFilter,0,MODE_SMA,PRICE_
   ArraySetAsSeries(CCI_Val,true);
   ArraySetAsSeries(MA_Val,true);
//---
   return(INIT_SUCCEEDED);
  }
```

```
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//---
   ArrayFree(CCI_Val);
   ArrayFree(MA_Val);
  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
//---

   CopyBuffer(CCI_Handle,0,0,3,CCI_Val);
   CopyBuffer(MA_Handle,0,0,3,MA_Val);
   double CurrentPrice=SymbolInfoDouble(_Symbol,SYMBOL_BII

   if(!PositionSelect(_Symbol))
     {
      if(CurrentPrice>MA_Val[1])
        {
         if(CCI_Val[1]>CCI_Upper)
           {
            if(CCI_Val[2]<CCI_Upper)
              {
               my_trade.Buy(0.1,_Symbol);
              }
           }
        }
      if(CurrentPrice<MA_Val[1])
        {
         if(CCI_Val[2]>CCI_Lower)
           {
            if(CCI_Val[1]<CCI_Lower)
              {
               my_trade.Sell(0.1,_Symbol);
              }
           }
        }
     }
   CopyBuffer(CCI_Handle,0,0,3,CCI_Val);

   if(PositionSelect(_Symbol))
     {
      if(my_position.PositionType()==POSITION_TYPE_BUY)
        {
         if(CCI_Val[1]<CCI_Upper_Exit)
           {
            my_trade.Sell(0.1,_Symbol);
           }
        }
```

```
      if(my_position.PositionType()==POSITION_TYPE_SELL)
        {
         if(CCI_Val[1]>CCI_Lower_Exit)
           {
            my_trade.Buy(0.1,_Symbol);
           }
        }
     }

  }
//+------------------------------------------------------------------+
```

Dabei ist besonders wichtig:

CTrade my_trade**;**

Aus dem CTrade Standardpaket. Durch diese Deklaration lässt sich auf Trades zugreifen. Man kann also Positionen einfach eröffnen und skalieren.

my_trade**.**Sell**(**0.1**,**_Symbol**);**

my_trade**.**Buy**(**0.1**,**_Symbol**);**

Des Weiteren ist der Abgleich der offenen Position ein wichtiger Schritt. Es wird geprüft, ob die offene Position eine Buy oder Sell Position ist.

my_position**.**PositionType**()==**POSITION_TYPE_SELL

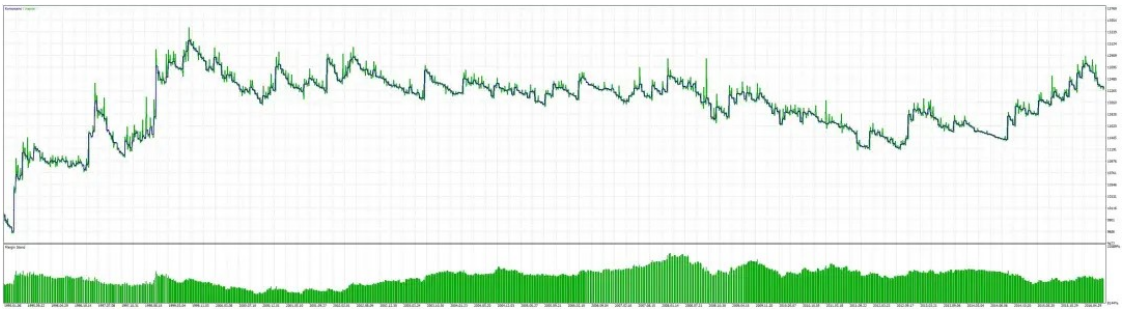
my_position**.**PositionType**()==**POSITION_TYPE_BUY

Das Array, welches die Indikatoren-Daten speichert, muss durch ArraySetSeries zu einer Zeitreihe modifiziert werden, welche dann mit der CopyBuffer Funktion befüllt wird. Danach kann in den folgenden Schritten auf die Werte der Zeitreihe entsprechend einer Array Abfrage mit [x] zugegriffen werden. So lassen sich über If Abfragen Signale generieren. Hier nochmals ein Beispiel aus dem CCI Code:

```
if(CurrentPrice<MA_Val[1])
        {
         if(CCI_Val[2]>CCI_Lower)
           {
            if(CCI_Val[1]<CCI_Lower)
              {
               my_trade.Sell(0.1,_Symbol);
              }
           }
        }
```

Dabei stellten CCI_Lower und Upper die Werte dar, ab denen wir Positionen eröffnen wollten. Hier wird zudem geprüft, ob der Wert gerade erst erreicht wird, ähnlich einem Cross Over, sodass ein Signal, im Falle eines Stop Outs nicht sofort wieder eine Position eröffnet. Das würde den Stop Out, also das Schließen der Position bei einem Gegensignal überflüssig machen.

Hier noch die Performance des EAs über die letzten 21 Jahre im USDJPY:



Wir hoffen, dass mit diesem Beispiel einiges zum Thema „Automatisiertes Handeln“ geklärt werden konnte. Wir werden noch ein Video zum Thema „System Tests und Optimierung“ erstellen. Um dieses nicht zu verpassen, meldet euch in unserem Newsletter an. Viele Grüße

[et_bloom_inline optin_id=optin_6]