MQL4 COURSE

By Coders' guru

www.forex-tsd.com

-14-

Your First Expert Advisor
Part 2

Welcome to the second part of creating Your First Expert Advisor lesson.
In the previous part we have taken the code generated by the ***new program wizard*** and added our own code which we are going to crack it today line by line.

Did you wear your ***coding gloves***? Let's crack.

Note: I have to repeat that our expert advisor is for educational purpose only and will not (or aimed to) make profits.

The code we have:

```
//+------------------------------------------------------------------+
//|                                                  My_First_EA.mq4 |
//|                                                      Coders Guru |
//|                                          http://www.forex-tsd.com |
//+------------------------------------------------------------------+
#property copyright "Coders Guru"
#property link      "http://www.forex-tsd.com"

//---- input parameters
extern double    TakeProfit=250.0;
extern double    Lots=0.1;
extern double    TrailingStop=35.0;
//+------------------------------------------------------------------+
//| expert initialization function                                   |
//+------------------------------------------------------------------+
int init()
  {
//----

//----
   return(0);
  }
//+------------------------------------------------------------------+
//| expert deinitialization function                                 |
//+------------------------------------------------------------------+
int deinit()
  {
//----
```

```
//----
   return(0);
  }

int Crossed (double line1 , double line2)
   {
      static int last_direction = 0;
      static int current_direction = 0;

      if(line1>line2)current_direction = 1; //up
      if(line1<line2)current_direction = 2; //down



      if(current_direction != last_direction) //changed
      {
            last_direction = current_direction;
            return (last_direction);
      }
      else
      {
            return (0);
      }
   }
//+------------------------------------------------------------------+
//| expert start function                                            |
//+------------------------------------------------------------------+
int start()
  {
//----


   int cnt, ticket, total;
   double shortEma, longEma;


   if(Bars<100)
     {
      Print("bars less than 100");
      return(0);
     }
   if(TakeProfit<10)
     {
      Print("TakeProfit less than 10");
      return(0);  // check TakeProfit
     }


   shortEma = iMA(NULL,0,8,0,MODE_EMA,PRICE_CLOSE,0);
   longEma = iMA(NULL,0,13,0,MODE_EMA,PRICE_CLOSE,0);

   int isCrossed  = Crossed (shortEma,longEma);

   total  = OrdersTotal();
   if(total < 1)
     {
       if(isCrossed == 1)
```

```
         {
ticket=OrderSend(Symbol(),OP_BUY,Lots,Ask,3,0,Ask+TakeProfit*Point,
"My EA",12345,0,Green);
            if(ticket>0)
              {
               if(OrderSelect(ticket,SELECT_BY_TICKET,MODE_TRADES))
Print("BUY order opened : ",OrderOpenPrice());
              }
            else Print("Error opening BUY order : ",GetLastError());
            return(0);
         }
         if(isCrossed == 2)
         {

            ticket=OrderSend(Symbol(),OP_SELL,Lots,Bid,3,0,
Bid-TakeProfit*Point,"My EA",12345,0,Red);
            if(ticket>0)
              {
               if(OrderSelect(ticket,SELECT_BY_TICKET,MODE_TRADES))
Print("SELL order opened : ",OrderOpenPrice());
              }
            else Print("Error opening SELL order : ",GetLastError());
            return(0);
         }
         return(0);
     }
   for(cnt=0;cnt<total;cnt++)
     {
      OrderSelect(cnt, SELECT_BY_POS, MODE_TRADES);
      if(OrderType()<=OP_SELL && OrderSymbol()==Symbol())
        {
         if(OrderType()==OP_BUY)   // long position is opened
           {
            // should it be closed?
           if(isCrossed == 2)
                {
                 OrderClose(OrderTicket(),OrderLots(),Bid,3,Violet);
// close position
                 return(0); // exit
                }
            // check for trailing stop
            if(TrailingStop>0)
              {
               if(Bid-OrderOpenPrice()>Point*TrailingStop)
                 {
                  if(OrderStopLoss()<Bid-Point*TrailingStop)
                    {
                     OrderModify(OrderTicket(),OrderOpenPrice(),Bid-
Point*TrailingStop,OrderTakeProfit(),0,Green);
                     return(0);
                    }
                 }
              }
           }
         else // go to short position
           {
```

```
            // should it be closed?
            if(isCrossed == 1)
              {
               OrderClose(OrderTicket(),OrderLots(),Ask,3,Violet);
// close position
               return(0); // exit
              }
            // check for trailing stop
            if(TrailingStop>0)
              {
               if((OrderOpenPrice()-Ask)>(Point*TrailingStop))
                 {
                  if((OrderStopLoss()>(Ask+Point*TrailingStop)) ||
(OrderStopLoss()==0))
                    {

OrderModify(OrderTicket(),OrderOpenPrice(),Ask+Point*TrailingStop,
OrderTakeProfit(),0,Red);
                     return(0);
                    }
                 }
              }
          }
        }
     }
   return(0);
  }
//+------------------------------------------------------------------+
```

The idea behind our expert advisor.

Before digging into cracking our code we have to explain the idea behind our expert advisor. Any expert advisor has to decide when to **enter** the market and when to **exit**. And the idea behind any expert advisor is what the entering and exiting **conditions** are? Our expert advisor is a simple one and the idea behind it is a simple too, let's see it.

We use two EMA indicators, the first one is the EMA of 8 days (sort EMA) and the second one is the EMA of 13 days (long EMA).

Note: using those EMAs or any thought in this lesson is not a recommendation of mine, they are for educational purpose only.

Entering (Open):

Our expert advisor will enter the market when the **short EMA** line crosses the **long EMA** line, the direction of each lines will determine the order type:
If the **short EMA** is **above** the **long EMA** will **buy** (long).
If the **short EMA** is **below** the **long EMA** we will **sell** (short).

We will open only **one** order at the same time.

Exiting (Close):

Our expert advisor will close the **buy** order when the **short EMA** crosses the **long EMA** and the **short EMA** is **below** the **long EMA**.
And will close the **sell** order when the **short EMA** crosses the **long EMA** and the **short EMA** is **above** the **long EMA**.

Our order (buy or sell) will automatically be closed too when the **Take profit** or the **Stop loss** points are reached.

Modifying:

Beside entering (opening) and exiting (closing) the market (positions) our expert advisor has the ability to modify existing positions based on the idea of **Trailing stop** point.
We will know how we implemented the idea of Trialing stop later in this lesson.

Now let's resume our code cracking.

```
//---- input parameters
extern double    TakeProfit=250.0;
extern double    Lots=0.1;
extern double    TrailingStop=35.0;
```

In the above lines we have asked the wizard to declare three **external** variables (which the user can set them from the properties window of our expert advisor).
The three variables are double data type. We have initialized them to default values (the user can change these values from the properties window, but it recommended to leave the defaults).

I have to pause again to tell you a little story about those variables.

Stop loss:

It's a limit point you set to your order when reached the order will be closed, this is useful to minimize your lose when the market going against you.
Stop losses points are always set below the current asking price on a buy or above the current bid price on a sell.

Trailing Stop

It's a kind of stop loss order that is set at a percentage level below (for a long position) or above (for a short position) the market price. The price is adjusted as the price fluctuates.
We will talk about this very important concept later in this lesson.

Take profit:

It's similar to stop lose in that it's a limit point you set to your order when reached the order will be closed

There are, however, two differences:

- There is no "trailing" point.
- The exit point must be set above the current market price, instead of below.

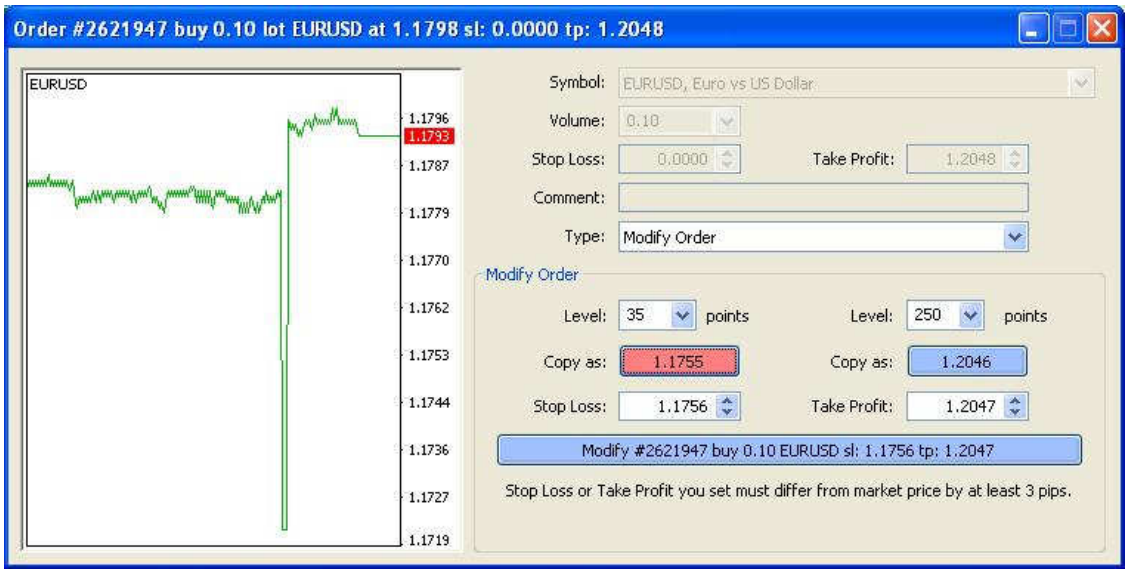

Figure 1 – Setting Stop loss and Take profit points

```
int Crossed (double line1 , double line2)
   {
      static int last_direction = 0;
      static int current_direction = 0;

      if(line1>line2)current_direction = 1; //up
      if(line1<line2)current_direction = 2; //down



      if(current_direction != last_direction) //changed
      {
            last_direction = current_direction;
            return (last_direction);
      }
      else
      {
            return (0);
      }

  }
```

As I told you before, the idea behind our expert advisor is monitor the crossing of the short EMA and the long EMA lines. And getting the direction of the crossing (which line is above and which line is below) which will determine the type of the order (buy, sell, buy-close and sell-close).

For this goal we have created the ***Crossed*** function.

The ***Crossed*** function takes two double values as parameters and returns an integer.
The first parameter is the value of the first line we want to monitor (the short EMA in our case) and the second parameter is the value of the second we want to (the long EMA).

The function will monitor the two lines every time we **call** it by saving the direction of the two lines in static variables to remember their state between the repeated calls.

- It will return **0** if there's no change happened in the last directions saved.
- It will return **1** if the direction has changed (the lines crossed each others) and the first line is above the second line.
- It will return **2** if the direction has changed (the lines crossed each others) and the first line is below the second line.

Note: You can use this function in your coming expert advisor to monitor any two lines and get the crossing direction.

Let's see how did we write it?

```
int Crossed (double line1 , double line2)
```

The above line is the function declaration, it means we want to create ***Crossed*** function which takes two double data type **parameters** and **returns** an integer.

When you **call** this function you have to pass to it two double parameters and it will return an integer to you.

You have to declare the function before using (calling) it. The place of the function doesn't matter, I placed it above **start()** function, but you can place it anywhere else.

```
static int last_direction = 0;
static int current_direction = 0;
```

Here we declared two static integers to hold the current and the last direction of the two lines. We are going to use these variables (they are static variables which means they save their value between the repeated calls) to check if there's a change in the direction of the lines or not.

we have initialized them to **0** because we don't want them to work in the first call to the function (if they worked in the first call the expert advisor will open an order as soon as we load it in the terminal).

```
if(current_direction != last_direction) //changed
```

In this line we compare the two static variables to check for changes between the last call of our function and the current call.
If ***last_direction*** not equal ***current_direction*** that's mean there's a change happened in the direction.

```
last_direction = current_direction;
return (last_direction);
```

In this case (***last_direction*** not equal ***current_direction)*** we have to reset our ***last_direction*** by assigning to it the value of the ***current_direction***.
And we will return the value of the ***last_direction***. This value will be **1** if the first line is above the second line and **2** if the first line is below the second line.

```
else
  {
     return (0);
  }
```

Else (***last_direction*** is equal ***current_direction)*** there's no change in the lines direction and we have to return **0**.

Our program will call this function in its **start()** function body and use the returned value to determine the appropriate action.

In the coming part of this lesson we will know how did we call the function, and we will know a lot about the very important trading functions.
To that day I hope you the best luck.

I welcome very much the questions and the suggestions.

See you
Coders' Guru
01-12-2005