MQL4 COURSE

By Coders' guru

www.forex-tsd.com

-12-

Your First Indicator
Part 3

Welcome to the third part of **"Your First Indicator"** lesson.
In the previous lesson we studied the code of our first indicator line by line and we reached the function **dinit()**.

I hope you've came from the previous lessons with a clear idea about what we have done.

Today we are going to study **start()** function and its content. And –***finally***- we will compile and run our first Indicator.

Are you ready? Let's hack the code line by line:

Our Code:

```
//+------------------------------------------------------------------+
//|                                            My_First_Indicator.mq4 |
//|                                                        Codersguru |
//|                                          http://www.forex-tsd.com |
//+------------------------------------------------------------------+
#property copyright "Codersguru"
#property link      "http://www.forex-tsd.com"

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_color1 Red

//---- buffers
double ExtMapBuffer1[];

//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int init()
  {
//---- indicators
   SetIndexStyle(0,DRAW_LINE);
   SetIndexBuffer(0,ExtMapBuffer1);
   string short_name = "Your first indicator is running!";
   IndicatorShortName(short_name);
//----
```

```
   return(1);
  }
//+------------------------------------------------------------------+
//| Custor indicator deinitialization function                       |
//+------------------------------------------------------------------+
int deinit()
  {
//----

//----
   return(0);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int start()
  {
   int    counted_bars=IndicatorCounted();

//---- check for possible errors
   if (counted_bars<0) return(-1);
//---- last counted bar will be recounted
   if (counted_bars>0) counted_bars--;

   int    pos=Bars-counted_bars;

   double dHigh , dLow , dResult;
   Comment("Hi! I'm here on the main chart windows!");

//---- main calculation loop
   while(pos>=0)
     {
         dHigh = High[pos];
         dLow = Low[pos];
         dResult = dHigh - dLow;
         ExtMapBuffer1[pos]= dResult ;
         pos--;
     }
//----
   return(0);
  }
//+------------------------------------------------------------------+
```

```
int start()
 {...
  return(0);
 }
```

As I told you before, we will spend 90% of programming life inside the braces of **start()** function. That's because it's the most important MQL4 Special functions.

On the contrary of the **init()** and **deinit** function, **start()** will not be called (by the terminal client) only one time, But every time a new quotation arrives to MetaTrader terminal client, every time the **start()** function has been called.

start() function returns an integer value like all of the MQL4 special function, where **0** means no error and any number else means an error has been occurred.

```
int    counted_bars=IndicatorCounted();
```

Here, we have defined the variable **counted_bars** as an integer type, and we have assigned to it the returned value of the function **IndicatorCounted()**.

int IndicatorCounted()

This function will return an integer type value holds the count of the **bars** which our indicator has been calculated them.

In the first launch of your indicator this count will be **0** because the indicator didn't calculate any bars yet. And after that it will be the count of total bars on the chart **-1**. (Please see the function **Bars** below).

```
if (counted_bars<0) return(-1);
```

We have got the number of **counted_bars** in the previous line of code by using **IndicatorCounted()** function.
This number must be 0 or greater if there's no errors have been occurred. If it's less than 0 that's means we have an error and we have to terminate the **start()** function using the return statement.

```
if (counted_bars>0) counted_bars--;
```

We are checking here if the **counted_bars** are greater than 0.
If that's true we decrease this number by subtracting 1 from it.
That's because we want to recount the last bar again.

We use the decrement operator (please review Lesson 4 - Operations & Expressions) for decreasing the value of **counted_bars** by **1**.

Note: We can write the expression counted_bars-- like this:

```
counted_bars = counted_bars-1;
```

```
int    pos=Bars-counted_bars;
```

Here, we are declaring the variable **pos** to hold the number of times our calculation loop

will work (see while loop later in this lesson). That's by subtracting the **counted_bars** from the count of total bars on the chart, we get the total bars count using **Bars()** function.
It's a good time to discuss **Bars()** function and its brother.

Pre-defined MQL4 variables:

Ask, Bid, Bars, Close, Open, High, Low, Time and Volume are functions although MQL4 called them Pre-defined variables. And I'll proof to you why they are functions.

Variable means a space in memory and data type you specify.
Function means do something and return some value, For example **Bars** collects and returns the number of the bars in chart. So, is it a variable?

Another example will proof for you that they are not variables:
If you type and compile this line of code:

```
Bars=1;
```

You will get this error: ***'Bars' - unexpected token***
That's because they are not variables hence you can't assign a value to them.

Another proof, the next line of code is a valid line and will not generate and error in compiling:

```
Alert(Bars(1));
```

You can't pass parameter to a variable, parameters passed only to the functions.

I'm so sorry for the lengthiness, let's discuss every function.

int Bars

This function returns an integer type value holds count of the total bars on the current chart.

double Ask

This function (used in your Expert Advisors) returns a double type value holds the buyer's price of the currency pair.

double Bid

This function (used in your Expert Advisor) returns a double type value holds the seller's price of the currency pair.

Note: For example, USD/JPY = 133.27/133.32 the left part is called the bid price (that is a price at which the trader sells), the second (the right part) is called the ask price (the price at which the trader buys the currency).

double Open[]

This function returns a double type value holds the opening price of the referenced bar. Where opening price is the price at the beginning of a trade period (year, month, day, week, hour etc)
For example: Open[0] will return the opening price of the current bar.

double Close[]

This function returns a double type value holds the closing price of the referenced bar. Where closing price is the price at the end of a trade period
For example: Close[0] will return the closing price of the current bar.

double High[]

This function returns a double type value holds the highest price of the referenced bar. Where it's the highest price from prices observed during a trade period.
For example: High [0] will return the highest price of the current bar.

double Low[]

This function returns a double type value holds the lowest price of the referenced bar. Where it's the lowest price from prices observed during a trade period.
For example: Low [0] will return the lowest price of the current bar.

double Volume[]

This function returns a double type value holds the average of the total amount of currency traded within a period of time, usually one day.
For example: Volume [0] will return this average for the current bar.

int Digits

This function returns an integer value holds number of digits after the decimal point (usually 4).

double Point

This function returns a double value holds point value of the current bar (usually 0.0001.

datetime Time[]

This function returns a datetime type value holds the open time of the referenced bar. For example: Time [0] will return the open time of the current bar.

```
double dHigh , dLow , dResult;
```

We declared three double type variables which we will use them later. Notice the way we used to declare the three of them at the same line by separating them by coma.

```
Comment("Hi! I'm here on the main chart windows!");
```

This line of code uses the **Comment** function to print the text “Hi! I'm here on the main chart windows!” on the left top corner of the main chart (figure1).
There are two similar functions:

void Comment(...)

This function takes the values passed to it (they can be any type) and print them on the left top corner of the chart (figure 1).
void Print (...)

This function takes the values passed to it (they can be any type) and print them to the expert log (figure 2).

void Alert(...)

This function takes the values passed to it (they can be any type) and display them in a dialog box (figure 3)

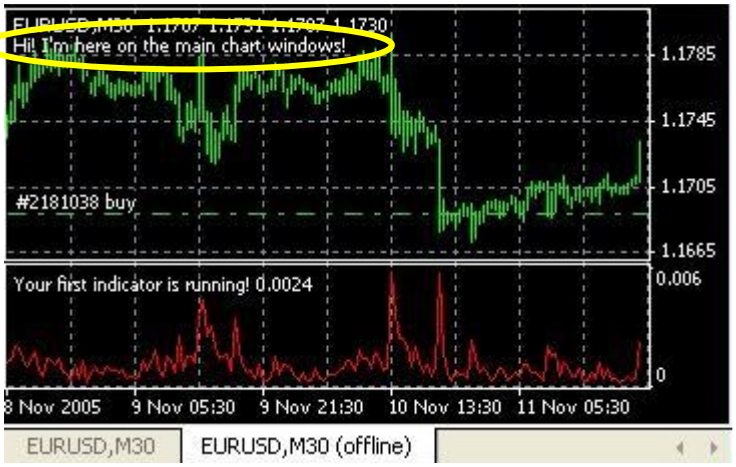

Figure 1 – Comment

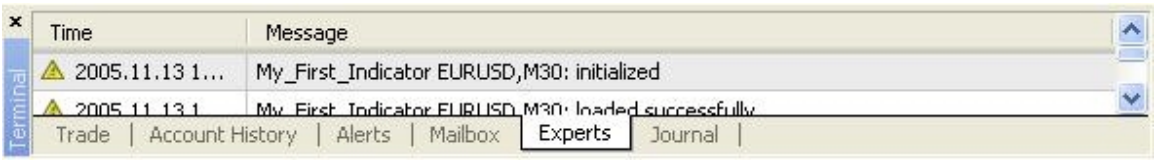

Figure 2- Expert log

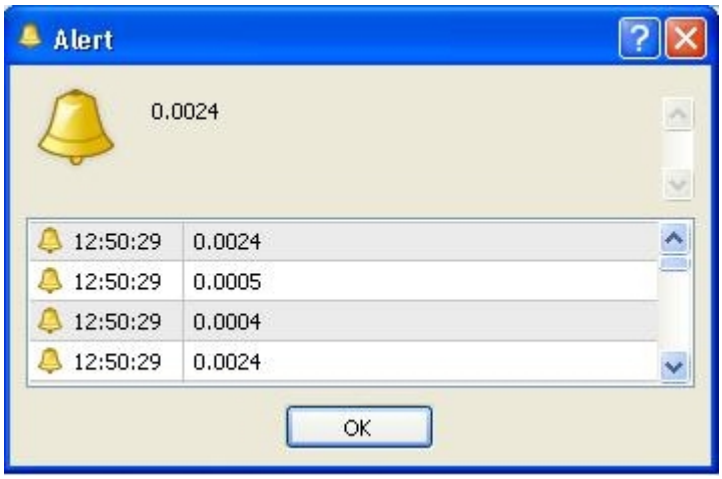

Figure 3 - Alerts

```
while(pos>=0)
     {
          dHigh = High[pos];
          dLow = Low[pos];
          dResult = dHigh - dLow;
          ExtMapBuffer1[pos]= dResult ;
          pos--;
     }
```

Now, it's the time to enter the loop for calculating our indicator points to draw them. Any value we assign to the array ExtMapBuffer1[] will be drawn on the chart (because we have assign this array to the drawn buffer using **SetIndexBuffer** function).

Before we enter the loop we have got the number of times the loop will work by subtracting the **counted_bars** from the total count of the bars on chart.
The number of times the loop will work called ***Loop variable*** which it's **pos** variable in our example.

We use the loop variable as a current bar of the calculation for example **High[pos]** will return the highest price of the pos bar.

In the loop body we assign to the variable **dHigh** the value of the highest price of the current loop variable.

And assign to the variable **dLow** the value of the lowest price of the current loop variable.
The result of subtracting **dLow** from **dHigh** will be assigned to the variable **dResult**.

Then we using the **dResult** to draw or indicator line, by assigning it to the drawn buffer array **ExtMapBuffer1[]**.

The last line of the loop is a decrement expression which will decrease the loop variable pos by 1 every time the loop runs. And when this variable reaches **-1** the loop will be terminated.

Finally, we can compile our indicator. Press **F5** or choose **Compile** from file menu. That will generate the executable file "My_First_indicator.ex4" which you can load in your terminal client.

To load your indicator click **F4** to bring the terminal client. Then From the **Navigator** window find the **My_First_indicator** and attach it to the chart (figure4).

Note: The indicator will not do much, but it believed that the subtraction of the highest and lowest of the price gives us the market's volatility.

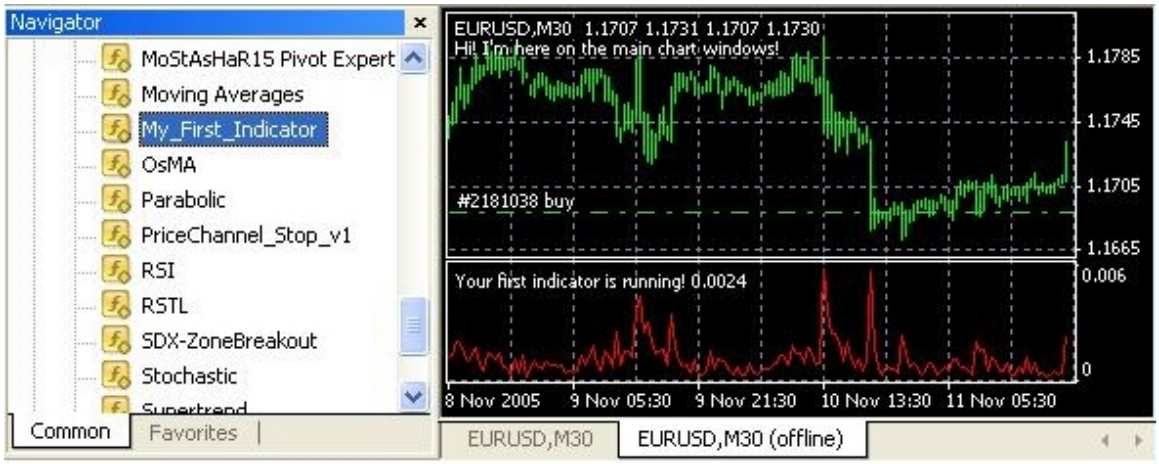

Figure 4 – My_First_Indicator

I hope you enjoyed your first indicator. And be prepared to your first Expert Advisor in the next lesson(s).

I welcome very much your questions and suggestions.

Coders' Guru
13-11-2005