Mql - Metatrader Development Course

Welcome to MQL4 course!

Welcome to the MQL4 course.
In this series, I will try to strip the mystique and confusion from MQL4 by giving you comprehensive tutorials with a straight forward example.

In this series of lessons, I will show you how to use the MQL4 for building your own Expert Advisors, Custom Indicators and Scripts.

If you are programming in C (or its superset C++) then you know a lot of MQL4 before even I start my lessons, if you didn't write in any programming language before, no problem, I'll guide you to understand the concept of programming in general as well.

So, let's start from the beginning.

MQL4? What, Why and Where?

MQL4 stands for **M**eta**Q**uotes **L**anguage **4.**
MetaQuotes is the company who built the MetaTrader Trading Platform.
And to make it stronger than the other trading platforms the company extended it by a built-in programming language that enables the user (you) to write his own trading strategies.

The language enables you to create one of the following:

1- **Expert Advisors**.
2- **Custom Indicators**.
3- **Scripts**.

• **Expert Advisor** is a program which can automate trading deals for you. For example it can automate your market orders, stops orders automatically, cancels/replaces orders and takes your profit.

• **Custom Indicator** is a program which enables you to use the functions of the technical indicators and it cannot automate your deals.

• **Script** is a program designed for single function execution. Unlike the Advisor, scripts are being held only once (on demand), and not by ticks. And of course has no access to indicator functions.

These were "What" MQL4 is? "Why" to use MQL4?
Now, "Where" do I write MQL4?

To write your MQL4 code and as anything else in world, you can choose one of two ways, the hard way and the easy way.

1- The hard way:

The hard way is using your favorite text editor and the command prompt to compile your program.
Notepad is not bad choice, but do not forget two things:

1- To save the file you have created in plain text format.
2- To save the file as .mp4 (that's to be easy to reopen it with MetaEditor), but you can save it as any extension you prefer.

After saving your program there is an extra step to make your code comes out to the light.
It's the Compiling step.
Compiling means to convert the human readable script that you have just wrote to the machine language that your computer understands.

MetaTrader has been shipped with its own compiler (the program which will convert your script to the machine language) called **MetaLang.exe**.

Metalang.exe is a console program which takes two parameters and output an **.ex4** file (the file which Metatrader understands).
The first parameter is "options" parameter and the only option available is **–q** quit
The second parameter is the full path to your .mql file.

The syntax will be in this format.
metalang [options…] filename

Example:

1- Find your metalang.exe path, it will be the same path of MetaTrader (here my path is D:\Program Files\MetaTrader 4).

2- Create a batch file and name it compile.bat (or any name you prefer).

3- Write these lines into the bat file then save it:
cd D:\Program Files\MetaTrader 4
metalang -q "D:\Program Files\MetaTrader 4\my_first_mql4_script.mq4"
(Don't forget to change the path to you MetaTrader installed path).

4- Run the batch file and if you are lucky person like me you will get a screen like figure 1.

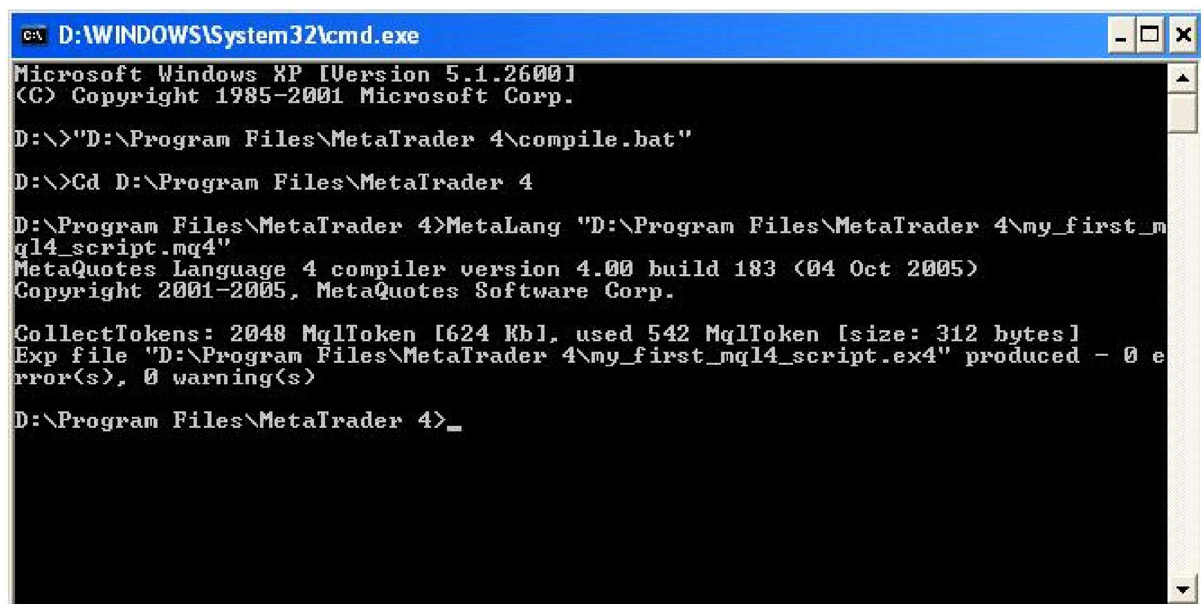


Figure 1 Metalang compiler

As you see you will get the output file "my_first_mql4_script.ex4"

2-The easy way:

Metatrader has been shipped with a good IDE (integrated development editor) called MetaEditor which has these features:

1- A text editor has the feature of highlighting different constructions of MQL4 language while you are writing/reading code.

2- Easy to compile your program, just click F5 and the MetaEditor will make all the hard work for you and produces the "ex4" file.
Besides it's easy to see what the wrong in your program is (in the Error Tab – see figure 2).

3- Built-in a dictionary book which you can access by highlight the keyword you want to know further about it then press F1.

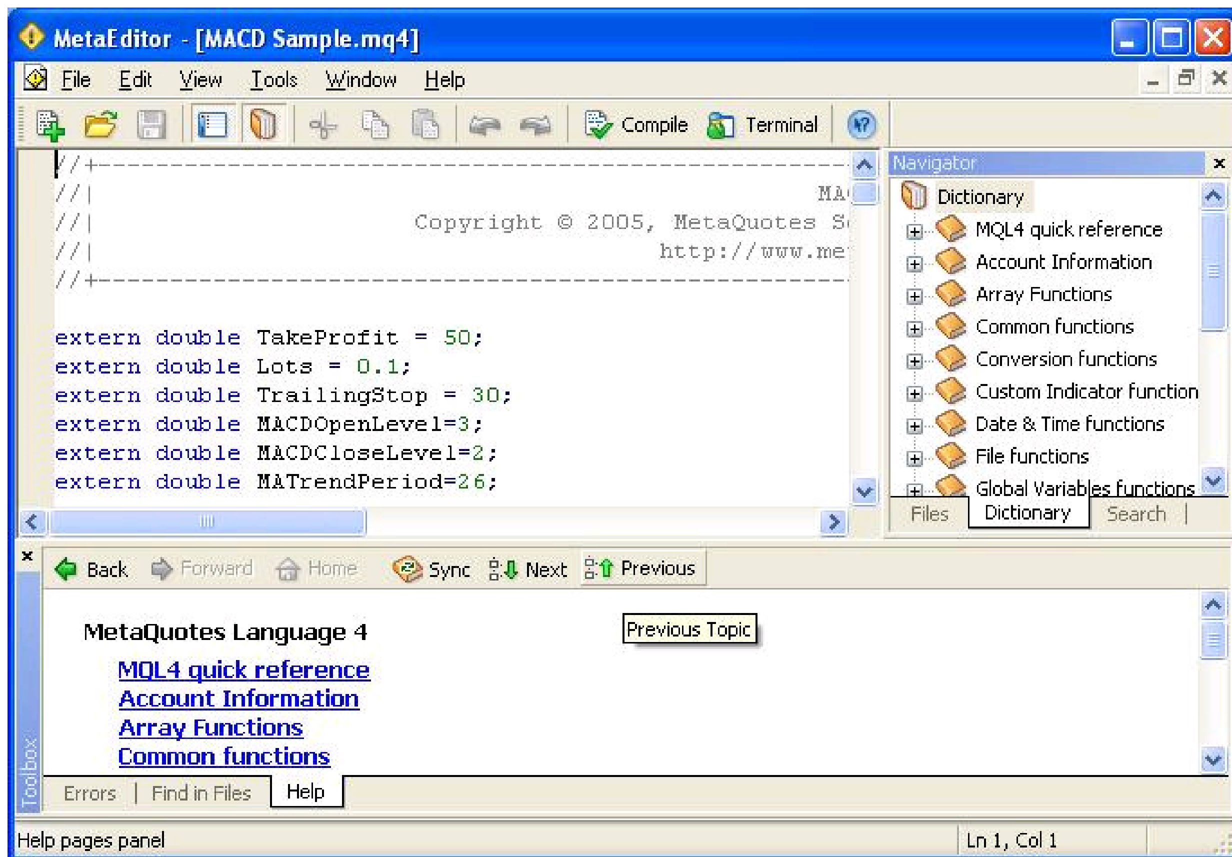


Figure 2 MetaEditor 4

In the coming lessons we will know more about MetaEditor.

Today I just came to say hello, tomorrow we will start the real works and will study the Syntax of MQL4.

I welcome very much the questions and the suggestions.

See you
Coders' Guru

Note: MetaTrader, the MetaTrader logo and MetaEditor are trademarks or registered trademarks of MetaQuotes Software Crop.

Lesson 2 - SYNTAX

Hi folks,

We are talking today about the SYNTAX rules of MQL4.

And as I told you before, *If you are programming in C (or its superset C++) then you know a lot of MQL4 before even I start my lessons.*

That's because the syntax of MQL4 is very like of the syntax of C.

The dictionary means of the word SYNTAX of a programming language is:

"The set of allowed reserved words and their parameters and the correct word order in the expression is called

*the syntax of language”. “**Wikipedia**”*

So, when we are studying the syntax of the language we are studying its grammar and writing rules which consist of:

- Format
- Comments
- Identifiers
- Reserved words

Let’s slice the cake.

1- Format:

When you write your code, you can freely use any set of spaces, tabs and empty lines you want to separate your code and your line of code to make them readable and eyes pleasing.

For example all of these lines are valid in MQL4:

```
double  MacdCurrent, MacdPrevious, SignalCurrent;


double

MacdCurrent,

MacdPrevious,

SignalCurrent;


double                    MacdCurrent,            MacdPrevious,           SignalCurrent;
```

But, as you see, the first line is more readable and easy to understand.

And as everything in the world there are exceptions to the rule:

1- You can't use new line in the "Controlling compilation"

You will know more about "Controlling compilation" in next lesson but just remember this is an exception.

For example the next line of code is invalid and the MQL4 compiler will complain:

```
#property
copyright "Copyright © 2004, MetaQuotes Software Corp."
```

This is the valid "Controlling compilation":

```
#property copyright "Copyright © 2004, MetaQuotes Software Corp."
```

2- You can't use new line or space in the middle of Constant values, Identifiers or Keywords.

For example this line is valid:

```
extern int MA_Period=13;
```

"extren" and "int" here are Keywords , "MA_Period" is an Identifier and "13" is a Constant value..

You will know more in the next lessons.

For example the next lines are invalids:

```
extern int MA_Period=1

3;
```

```
extern int MA_Period=1		3;
```

Notice the tab between 1 and 3.

```
ex

tern int MA_Period=13;
```

2- Comments:

To make the programming world easier, any programming language has its style of writing comments.

You use Comments to write lines in your code which the compiler will ignore then but it clears your code and makes it understandable.

Assume that you write a program in the summer and in the winter you want to read it. Without comments -even you are the code's creator- you can't understand all these puzzled lines.

MQL4 (& C/C++) uses two kinds of comments styles:

1- Single line comments

The Single line comment starts with "//" and ends with the new line.

For example:

```
//This is a comment

extern int MA_Period=13;
```

```
extern int MA_Period=13; //This is another comment
```

2- Multi-line comments

The multi-line comment start with “/*” and ends with “*/”.

And you can comment more than line or more by putting “/*” at the start of the first line, and “*/” at the end of the last line.

For example:

```
/* this

is

multi

line

comment*/
```

You can also nest single line comment inside multi lines comment like that:

```
/* this

is

multi       //another comment nested here.

line

comment*/
```

This is a valid comment too:

```
extern int /*HELLO! I'm a comment*/ MA_Period=13;
```

But this is invalid comment:

```
extern int //test MA_Period=13;
```

3- Identifiers:

An identifier is the name you choose to your variables, constants and functions.

For example MA_Period here is an identifier:

```
extern int MA_Period=13;
```

There are few rules and restrictions for choosing those names:

1- The length of the Identifier must not exceed **31** characters.

2- The Identifier must begin with a letter (capital or small) or the underlining symbol _.
So, it can't be started with a number or another symbol except the underlining symbol.

3- You can't use any reserved words as an Identifier.

You will see the list of the reserved words too soon.

4- The identifiers' names are case sensitive.

So, **MA_PERIOD** not the same as **ma_period** or **MA_Period**

Let's take some examples:

Name1	Valid
_Name1	Valid
1Name	Invalid (don't start with number)
~Name1	Invalid (you can only use underline symbol)
N~ame1	Invalid (you can only use underline symbol)
i_love_my_country_and_my_country_loves_all_the_world	Invalid (you can't exceed the 31 characters length)
Color	Valid
color	Invalid (you can't use reversed word, and **color** is one of them)

4- Reserved words:

There are "words" which the language uses them for specific actions.

So, they are reserved to the language usage and you can't use them as an identifier name or for any other purpose.

This is the list of the reserved words (*from the MQL4 guide*):

Data types	**Memory classes**	**Operators**	**Other**
bool	extern	Break	false
color	static	Case	true
datetime		continue	
double		Default	

int	Else
string	For
void	If
	Return
	Switch
	While

For example the next lines of code are invalid:

extern int **datetime** =13;

int **extern** =20;

double **continue** = 0;

Lesson 3 - MQL4 Data types

What's the Data type mean?

Any programming language has a set of names of the memory representation of the data.

For example if the memory holds numbers between -2147483648 to 2147483647, the most of the programming languages will name this data as "**Integer**" data type.

Variables?

Variables are the names that refer to sections of memory into which data can be stored.

To help you think of this as a picture, imagine that memory is a series of different size boxes. The box size is memory storage area required in bytes.

- In order to use a box to store data, the box must be given a name; this process is known as **declaration.**
- In the declaration process you use a word tell the computer what's the kind and size of the box you want to use, this word known as **keyword**.
- It helps if you give a box a meaningful name that relates to the type of information which make it easier to find the data, this name is the **variable constant**.
- Data is placed into a box by **assigning** the data to the box.
- When we set the value of the box you have created in the same line you declared the variable; this process is known as **initialization.**

When we create a variable we are telling the computer that we want him to assign a specified memory length (in bytes) to our variable, since storing a simple number, a letter or a large number is not going to occupy the same space in memory, so the computer will ask us what's the kind of data and how much the length of the data? That is the Data type for.

For example if we said this line of code to the computer:

```
int  MyVaraible=0;
```

That's mean we are asking the computer to set a block of 4 bytes length to our variable named "MyVaraiable".

In the previous example we have used:

int ß Keyword

int ß Integer data type.

int ß Declaration

MyVaraible ß Variable's constant.

=0 ß Initialization

We will know more about variables in a coming lesson.

In MQL4, these are the kinds of Data types:

- Integer (int)
- Boolean (bool)
- Character (char)
- String (string)
- Floating-point number (double)
- Color (color)
- Datetime (datetime)

1- Integer

An integer, is a number that can start with a + or a - sign and is made of digits. And its range value is between -2147483648 to 2147483647.

MQL4 presents the integer in decimal or hexadecimal format.

For example the next numbers are Integers:

12, 3, 2134, 0, -230

0x0A, 0x12, 0X12, 0x2f, 0xA3, 0Xa3, 0X7C7

We use the keyword **int** to create an integer variable.

For example:

```
int intInteger = 0;

int intAnotherIntger = -100;

int intHexIntger=0x12;
```

Decimal and Hexadecimal:

Decimal notation is the writing of numbers in the base of 10, and uses digits (0, 1, 2, 3, 4, 5, 6, 7, 8 and 9) to represent numbers. These digits are frequently used with a decimal point which indicates the start of a fractional part, and with one of the sign symbols + (plus) or − (minus) to indicate sign.

Hexadecimal is a numeral system with a base of 16 usually written using the symbols 0–9 and A–F or a–f.

For example, the decimal numeral 79 can be written as 4F in hexadecimal.

2- Boolean

Boolean variable is a data type which can hold only two values, true and false (or their numeric representation, 0 and 1). And it occupies 1 bit of the memory.

In MQL4, false,FALSE,False,true,TRUE and True are equals.

Boolean named like this in the honor of the great mathematician Boole George.

We use the keyword **bool** to create a boolean variable.

For example:

```
bool I = true;

bool bFlag = 1;

bool bBool=FALSE;
```

3- Character

MQL4 names this Data type "Literal".

A character is one of 256 defined alphabetic, numeric, and special key elements

defined in the ASCII (American Standard Code for Information Interchange) set.

Characters have integer values corresponding to location in the ASCII set.

You write the character constant by using single quotes (') surrounding the character.

For example:

```
'a' , '$' , 'Z'
```

We use the keyword **int** to create a character variable.

For example:

```
int chrA = 'A';

int chrB = '$';
```

Some characters called Special Characters can't present directly inside the single quotes because they have a reserved meanings in MQL4 language.

Here we use something called **Escape Sequence** to present those special characters,

And that by prefixing the character with the backslash character (\).

For example:

```
int chrA = '\\';      //slash character

int chrB = '\n';      //new line
```

This is the list of Escape Sequence characters used in MQL4.

carriage return	\r
new line	\n
horizontal tab	\t
reverse slash	\\
single quote	\'
double quote	\"
hexadecimal ASCII-code	\xhh

4- String

The string data type is an array of characters enclosed in double quote (").

The array of characters is an array which holds one character after another, starting at index 0. After the last character of data, a NULL character is placed in the next array location. It does not matter if there are unused array locations after that.

A NULL character is a special character (represented by the ASCII code 0) used to mark the end of this type of string.

See figure 1 for a simple representation of the string constant "hello" in the characters array.

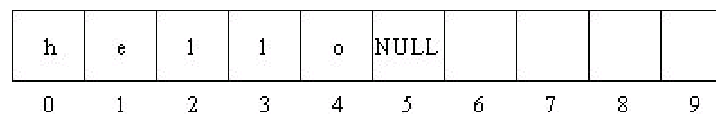


Figure 1 – Characters array

MQL4 limits the size of the string variable to 255 characters and any character above 255 characters will generate this error: (*too long string (255 characters maximum)*).

You can use any special character -mentioned above- in your string constant by prefixing it with the backslash (\).

We use the keyword string to create a string variable.

For example:

```
string str1 = "Hello world1, with you coders guru”;

string str2 = "Copyright © 2005, \"Forex-tsd forum\".";  //Notice the use of (") character.

string str3 = "1234567890";
```

5- Floating-point number (double)

Floating point number is the Real Number (that is, a number that can contain a fractional part beside the integer part separated with (.) dot).Ex: 3.0,-115.5, 15 and 0.0001.

And its range value is between 2.2e-308 to 1.8e308.

We use the keyword double to create a floating-point variable.

For example:

```
double dblNumber1 = 1000000000000000;

double dblNumber3 = 1/4;

double dblNumber3 = 5.75;
```

6- Color

Color data type is a special MQL4 data type, which holds a color appears on the MetaTrader chart when you create your own Expert Advisor or Custom Indictor and the user can change it from the property tab of your Expert Advisor or Custom Indictor.

You can set the Color variable constant in three ways:

1- By the color name: For the well know colors (called Web Colors Set) you can assign the name of the color to the color variable, see the list of the Web Colors Set.

2- By Character representation (MQL4 named it this name): In this method you use the keyword (C) followed by two signal quotations ('). Between the two signal quotations you set the value of the red, green and blue (know as RGB value of the color). These values have to be between: 0 to 255. And you can write these values in decimal or hexadecimal format.

3- By the integer value: Every color in the Web Colors Set has its integer value which you can write it in decimal or hexadecimal format. And you can assign the Integer value of the color to the color variable. The hexadecimal color format looks like this: 0xBBGGRR where BB is the blue value, GG is green value and RR is the red value.

For example:

```
// symbol constants
```

```
C'128,128,128'    // gray

C'0x00,0x00,0xFF' // blue

// named color

Red

Yellow

Black

// integer-valued representation

0xFFFFFF          // white

16777215          // white

0x008000          // green

32768             // green
```

We use the keyword color to create a color variable.

For example:

```
color clr1= Red;

color clr1= C'128,128,128' ;

color clr1=32768;
```

Web Colors Set

Black	DarkGreen	DarkSlateGray	Olive	Green	Teal	
Maroon	Indigo	MidnightBlue	DarkBlue	DarkOliveGreen	SaddleBrown	
SeaGreen	DarkGoldenrod	DarkSlateBlue	Sienna	MediumBlue	Brown	I
LightSeaGreen	DarkViolet	FireBrick	MediumVioletRed	MediumSeaGreen	Chocolate	
Goldenrod	MediumSpringGreen	LawnGreen	CadetBlue	DarkOrchid	YellowGreen	
DarkOrange	Orange	Gold	Yellow	Chartreuse	Lime	
DeepSkyBlue	Blue	Magenta	Red	Gray	SlateGray	
LightSlateGray	DeepPink	MediumTurquoise	DodgerBlue	Turquoise	RoyalBlue	
IndianRed	MediumOrchid	GreenYellow	MediumAquamarine	DarkSeaGreen	Tomato	
MediumPurple	PaleVioletRed	Coral	CornflowerBlue	DarkGray	SandyBrown	M

DarkSalmon	BurlyWood	HotPink	Salmon	Violet	LightCoral	
Plum	Khaki	LightGreen	Aquamarine	Silver	LightSkyBlue	I
PaleGreen	Thistle	PowderBlue	PaleGoldenrod	PaleTurquoise	LightGrey	
Moccasin	LightPink	Gainsboro	PeachPuff	Pink	Bisque	L
LemonChiffon	Beige	AntiqueWhite	PapayaWhip	Cornsilk	LightYellow	
Lavender	MistyRose	OldLace	WhiteSmoke	Seashell	Ivory	
LavenderBlush	MintCream	Snow	White			

7- Datetime

Datetime data type is a special MQL4 data type, which holds a date and time data. You set the Datetime variable by using the keyword (D) followed by two signal quotations ('). Between the two signal quotations you write a character line consisting of 6 parts for value of year, month, date, hour, minutes, and seconds. Datetime constant can vary from Jan 1, 1970 to Dec 31, 2037.

For example:

```
D'2004.01.01 00:00'     // New Year

D'1980.07.19 12:30:27'

D'19.07.1980 12:30:27'

D'19.07.1980 12'        //equal to D'1980.07.19 12:00:00'

D'01.01.2004'           //equal to D'01.01.2004 00:00:00'
```

We use the keyword datetime to create a datetime variable.

For example:

```
datetime dtMyBirthDay= D'1972.10.19 12:00:00';

datetime dt1= D'2005.10.22 04:30:00';
```

Lesson 4 - MQL4 Operations & Expressions

What's the meaning of Operations & Expressions?

You know the operations very well. If I told you that (+,-,*, /) are the basic arithmetical operators, you will remember very fast what's the operator means.

I hear you saying "OK, I know the operations; could you tell me what's the meaning of the expression?"

Identifiers (do you remember them? If not, Review the SYNTAX lesson) together with the Operations produce the Expressions.

Puzzled? Let's illustrate it in an example:

```
x = (y*z)/w;
```

x,y,z and w, here are identifiers.

=,* and / are the operators.

The whole line is an expression.

When the expressions combined together it makes a statement.

And when the statements combined together it makes a function and when the functions combined together it makes a program.

In the remaining of this lesson we are going to talk about the kinds operators used in MQL4.

So, let's start with the basic arithmetical operators:

1- Arithmetical operators:

In MQL4 there are 9 Arithmetical operations

This is the list of them with the usage of each:

Operator	Name	Example	Description
+	Addition operator	**A = B + C;**	Add **A** to **B** and assign the result to **C**.
-	Subtraction operator	**A = B - C;**	Subtract **C** from **B** and assign the result to **C**.
+ -	Sign changer operators	**A = -A;**	Change the sign of **A** from positive to negative.
*	Multiplication operator	**A = B * C;**	Multiply **B** and **C** and assign the result to **A**.
/	Division operator	**A = B / C;**	Divide **B** on **C** and assign the result to **A**.
%	Modulus operator	**A =A % C;**	**A** is the reminder of division of **B** on **C**. (ex: 10%2 will produce 0, 10%3 will produce 1).
++	Increment operator	**A++;**	Increase **A** by **1** (ex: if A =1 make it 2).
--	Decrement operator	**A--;**	Decrease **1** from **A** (ex: if A =2 make it 1).

Note: You can't combine the increment and decrement operator with other expressions.

For example you can't say:

```
A=(B++)*5;
```

But you can write it like that:

```
A++;

B=A*5;
```

2- Assignment operators:

The purpose of any expression is producing a result and the assignment operators setting the left operand with this result.

For example:

A = B * C;

Here we multiply B and C and assign the result to A.

(=) here is the assignment operator.

In MQL4 there are 11 assignments operations

This is the list of them with the usage of each:

Operator	Name	Example	Description
=	Assignment operator	**A = B;**	Assign **B** to **A.**
+=	Additive Assignment operator	**A += B;**	It's equal to: **A = A + B;** Add **B** to **A** and assign the result to **A**.
-=	Subtractive Assignment operators	**A -= B;**	It's equal to: **A = A - B;** Subtract B from A and assign the result to A.
*=	Multiplicative Assignment operator	**A *= B;**	It's equal to: **A = A * B;** Multiply A and B and assign the result to A.
/=	Divisional Assignment operator	**A /= B;**	It's equal to: **A = A / B;** Divide A on B and assign the result to A.
%=	Modulating Assignment operator	**A %= B;**	It's equal to: **A = A % B;** Get the reminder of division of **A** on **B** and assign the result to **A**.
>>=	Left Shift Assignment operator	**A >>= B;**	It shifts the bits of **A** left by the number of bits specified in **B**.
<<=	Right Shift Assignment operator	**A <<= B;**	It shifts the bits of **A** right by the number of bits specified in **B**.
&=	AND Assignment operator	**A &= B;**	Looks at the binary representation of the values of **A** and **B** and does a bitwise AND operation on them.
|=	OR Assignment operator	**A |= B;**	Looks at the binary representation of the values of **A** and **B** and does a bitwise OR operation on them.
^=	XOR Assignment operator	**A ^= B;**	Looks at the binary representation of the values of two **A** and **B** and does a bitwise exclusive OR (XOR) operation on them.

3- Relational operators:

The relational operators compare two values (operands) and result false or true only.

It's is like the question "Is John taller than Alfred? Yes / no?"

The result will be false only if the expression produce zero and true if it produces any number differing from zero;

For example:

```
4 == 4;          //true
4 < 4;           //false
4 <= 4          //true;
```

In MQL4 there are 6 Relational operations

This is the list of them with the usage of each:

Operator	Name	Example	Description
==	Equal operator	**A == Bi;**	True if **A** equals **B** else False.
!=	Not Equal operator	**A != B;**	True if **A** does not equal **B** else False.
<	Less Than operators	**A < B;**	True if **A** is less than **B** else False.
>	Greater Than operator	**A > B;**	True if **A** is greater than **B** else False.
<=	Less Than or Equal operator	**A <= B;**	True if **A** is less than or equals **B** else False.
>=	Greater Than or Equal operator	**A >= B;**	True if **A** is greater than or equals **B** else False.

4- Logical operators:

Logical operators are generally derived from *Boolean algebra*, which is a mathematical way of manipulating the *truth values* of concepts in an abstract way without bothering about what the concepts actually *mean*. The truth value of a concept in Boolean value can have just one of two possible values: true

or false.

MQL4 names the Logical operators as Boolean operators

MQL4 uses the most important 3 logical operators.

This is the list of them with the usage of each:

<table>
<tr><th>Operator</th><th>Name</th><th>Example</th><th>Description</th></tr>
<tr><td>&&</td><td>AND operator</td><td>A && B;</td><td>If either of the values are zero the value of the expression is zero, otherwise the value of the expression is 1. If the left hand value is zero, then the right hand value is not considered.</td></tr>
<tr><td>||</td><td>OR operator</td><td>A || B;</td><td>If both of the values are zero then the value of the expression is 0 otherwise the value of the expression is 1. If the left hand value is non-zero, then the right hand value is not considered.</td></tr>
<tr><td>!</td><td>NOT operator</td><td>!A;</td><td>Not operator is applied to a non-zero value then the value is zero, if it is applied to a zero value, the value is 1.</td></tr>
</table>

5- Bitwise operators:

The bitwise operators are similar to the logical operators, except that they work on a smaller scale -- binary representations of data.

The following operators are available in MQL4:

<table>
<tr><th>Operator</th><th>Name</th><th>Example</th><th>Description</th></tr>
<tr><td>&</td><td>AND operator</td><td>A & B;</td><td>Compares two bits and generates a result of 1 if both bits are 1; otherwise, it returns 0.</td></tr>
<tr><td>|</td><td>OR operator</td><td>A | B;</td><td>Compares two bits and generates a result of 1 if the bits are complementary; otherwise, it returns 0.</td></tr>
<tr><td>^</td><td>EXCLUSIVE-OR operator</td><td>A ^ B;</td><td>Compares two bits and generates a result of 1 if either or both bits are 1; otherwise, it returns 0.</td></tr>
<tr><td>~</td><td>COMPLEMENT operator</td><td>~A;</td><td>Used to invert all of the bits of the operand.</td></tr>
</table>

>>	The SHIFT RIGHT operator	**A >> B;**	Moves the bits to the right, discards the far right bit, and assigns the leftmost bit a value of 0. Each move to the right effectively divides op1 in half.
<<	The SHIFT LEFT operator	**A << B;**	Moves the bits to the left, discards the far left bit, and assigns the rightmost bit a value of 0. Each move to the left effectively multiplies op1 by 2.

Note Both operands associated with the bitwise operator must be integers.

6- Other operators:

There are some operators which used in MQL4 and don't belong to one of the previous categories:

1- The array indexing operator ([]).

2- The function call operator (());

3- The function arguments separator operator -comma (,)

We will know more about the Arrays and Functions in the next lessons, so just remember these 3 operators as "Other operators".

Operators Precedence:

If you don't explicitly indicate the order in which you want the operations in a compound expression to be performed, the order is determined by the *precedence* assigned to the operators in use within the expression. Operators with a higher precedence get evaluated first. For example, the division operator has a higher precedence than does the addition operator. Thus, the two following statements are equivalent:

```
x + y / 100

x + (y / 100) //unambiguous, recommended
```

When writing compound expressions, you should be explicit and indicate with parentheses () which operators should be evaluated first. This practice will make your code easier to read and to maintain.

The following table shows the precedence assigned to the operators in the MQL4. The operators in this table are listed in precedence order: The higher in the table an operator appears, the higher its precedence. Operators with higher precedence are evaluated before operators with a relatively lower precedence. Operators on the same group have equal precedence. When operators of equal precedence appear in the same expression, a rule must govern which is evaluated first. All binary operators except for the assignment operators are evaluated from left to right. Assignment operators are evaluated right to left.

()	Function call	*From left to right*
[]	Array element selection	
!	Negation	*From left to right*
~	Bitwise negation	
-	Sign changing operation	
*	Multiplication	*From left to right*
/	Division	
%	Module division	
+	Addition	*From left to right*
-	Subtraction	
<<	Left shift	*From left to right*
>>	Right shift	
<	Less than	*From left to right*

<= **Less than or equals**

> **Greater than**

>= **Greater than or equals**

== **Equals** ***From left to right***

!= **Not equal**

& **Bitwise AND operation** ***From left to right***

^ **Bitwise exclusive OR** ***From left to right***

&& **Logical AND** ***From left to right***

|| **Logical OR** ***From left to right***

=	Assignment	*From right to left*
+=	Assignment addition	
-=	Assignment subtraction	
*=	Assignment multiplication	
/=	Assignment division	
%=	Assignment module	
>>=	Assignment right shift	
<<=	Assignment left shift	
&=	Assignment bitwise AND	
|=	Assignment bitwise OR	
^=	Assignment exclusive OR	
,	Comma	*From left to right*

Lesson 5 - Loops & Decisions (Part1)

Welcome to the fifth lesson in my course about MQL4.

The normal flow control of the program you write in MQL4 (And in others languages as well) executes from top to bottom, A statement by a statement.

A statement is a line of code telling the computer to do something.

For example:

Print("Hello World");

return 0;

A semicolon at end of the statement is a crucial part of the syntax but usually easy to forget, and that's make it the source of 90% of errors.

But the top bottom execution is not the only case and it has two exceptions,

They are the loops and the decisions.

The programs you write like -the human- decides what to do in response of circumstances changing. In these cases the flow of control jumps from one part of the program to another.

Statements cause such jumps is called Control Statements.

Such controls consist of Loops and Decisions.

LOOPS

Loops causing a section of your program to be repeated a certain number of times.

And this repetition continues while some condition is true and ends when it becomes false.

When the loop end it passes the control to next statement follow the loop section.

In MQL4 there are two kinds of loops:

The for Loop

The for loop considered the easiest loop because all of its control elements are gathered in one place.

The for loop executes a section of code a fixed number of times.

For example:

```
int j;

for(j=0; j<15; j++)

            Print(j);
```

How does this work?

The for statement consists of for keyword, followed by parentheses that contain three expressions separated by semicolons:

for(j=0; j<15; j++)

These three expressions are the initialization expression, the test expression and the increment expression:

j=0 ß initialization expression

j<15 ß test expression

J++ ß increment expression

The body of the loop is the code to be executed the fixed number of the loop:

Print(j);

This executes the body of the loop in our example for 15 times.

Note: the for statement in not followed by a semicolon. That's because the for statement and the loop body are together considered to be a program statement.

The initialization expression:

The initialization expression is executed only once, when the loop first starts. And its purpose to give the loop variable an initial value (0 in our example).

You can declare the loop variable outside (before) the loop like our example:

int j;

Or you can make the declaration inside the loop parentheses like this:

for(int j=0; j<15; j++)

The previous two lines of code are equal, except the Scope of each variable (you will know more about the variable declaration and scopes in the Variables lesson).

The outside declaration method makes every line in the code block to know about the variable, while the inside declaration makes only the for loop to know about the variable.

You can use more that one initialization expression in for loop by separating them with comma (,) like this:

```
int i;

int j;

for(i=0 ,j=0;i<15;i++)

  Print(i);
```

The Test expression:

The test expression always a relational expression that uses relational operators (please refer to relational operators in the previous lesson).

It evaluated by the loop every time the loop executed to determine if the loop will continue or will stop. It will continue if the result of the expression is true and will stop if it false.

In our example the body loop will continue printing i (Print(i)) while the case j<15 is true. For example the

j = 0,1,2,3,4,5,6,7,8,9,10,11,12,13 and 14.

And when j reaches 15 the loop will stops and the control passes to the statement following the loop.

The Increment expression:

The increment expression changes the value of the loop variable (j in our example) by increase it value by 1.

It executed as the last step in the loop steps, after initializing the loop variable, testing the test expression and executing the body of the loop.

Figure 1 shows a flow chart of the for loop.

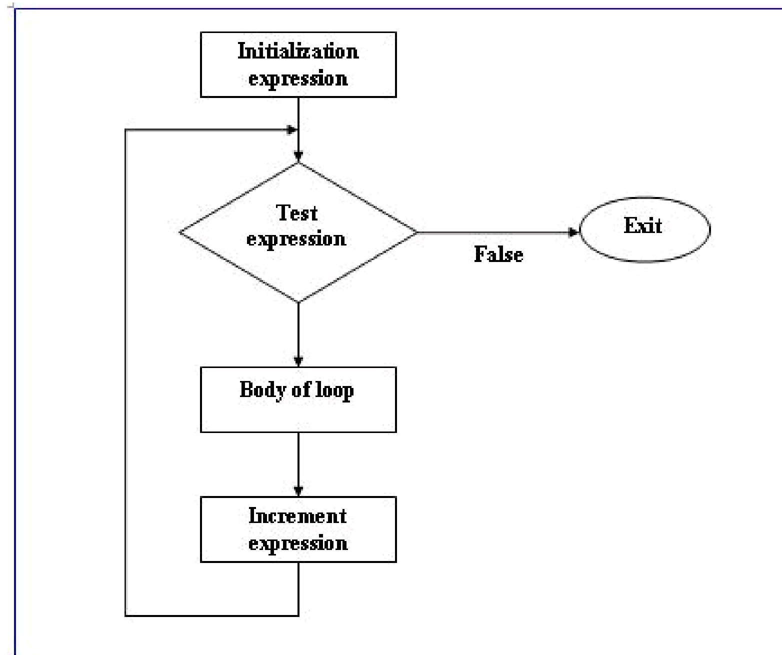


Figure 1 - Flow chart of the for loop

Like the initialization expression, in the increment expression you can use more than one increment expression in the for loop by separating them with comma (,) like this:

```
int i;
```

```
int j;

for(i=0 ,j=0;i<15,i<;i++,j++)

   Print(i);
```

But you can only use one test expression.

Another notice about the increment expression, it’s not only can increase the variable of the loop, but it can perform and operation it like for example decrements the loop variable like this:

```
int i;

for(i=15;i>0,i<;i--)

   Print(i);
```

The above example will initialize the i to 15 and start the loop, every time it decreases i by 1 and check the test expression (i>0).

The program will produce these results: 15,14,13,12,11,10,9,8,7,6,5,4,3,2,1.

Multi statement in the loop body:

In our previous examples, we used only one statement in the body of the loop, this is not always the case.

You can use multi statements in the loop body delimited by braces like this:

```
for(int i=1;i<=15;i++)

{

   Print(i);

   PlaySound("alert.wav");

}
```

In the above code the body of the loop contains two statements, the program will execute the first

statement then the second one every time the loop executed.

Don’t forget to put a semicolon at the end of every statement.

The Break Statement:

When the keyword presents in the for loop (and in while loop and switch statement as well) the execution of the loop will terminate and the control passes to the statement followed the loop section.

For example:

```
for(int i=0;i<15;i++)

{

  if((i==10)

    break;

  Print(i);

}
```

The above example will execute the loop until i reaches 10, in that case the break keyword will terminate the loop. The code will produce these values: 0,1,2,3,4,5,6,7,8,9.

The Continue Statement:

The break statement takes you out the loop, while the continue statement will get you back to the top of the loop (parentheses).

For example:

```
for(int i=0;i<15; i++)

{

   if(i==10) continue;

   Print(i)
```

```
}
```

The above example will execute the loop until i reaches 10, in that case the continue keyword will get the loop back to the top of the loop without printing i the tenth time. The code will produce these values: 0,1,2,3,4,5,6,7,8,9,11,12,13,14.

Latest note:

You can leave out some or all of the expressions in for loop if you want, for example:

for(;;)

This loop is like while loop with a test expression always set to true.

We will introduce the while loop to you right now.

The while Loop

The for loop usually used in the case you know how many times the loop will be executed. What happen if you don't know how many times you want to execute the loop?

This the while loop is for.

The while loop like the for loop has a Test expression. But it hasn't Initialization or Increment expressions.

This is an example:

```
int i=0;

while(i<15)

{

  Print(i);
```

```
    i++;
  }
```

In the example you will notice the followings:

- **The loop variable had declared and initialized before the loop, you can not declare or initialize it inside the parentheses of the while loop like the for loop.**
- **The i++ statement here is not the increment expression as you may think, but the body of the loop must contain some statement that changes the loop variable, otherwise the loop would never end.**

How the above example does work?

The while statement contains only the Test expression, and it will examine it every loop, if it's true the loop will continue, if it's false the loop will end and the control passes to the statement followed the loop section.

In the example the loop will execute till i reaches 16 in this case i<15=false and the loop ends.

Figure 2 shows a flow chart of the while loop.

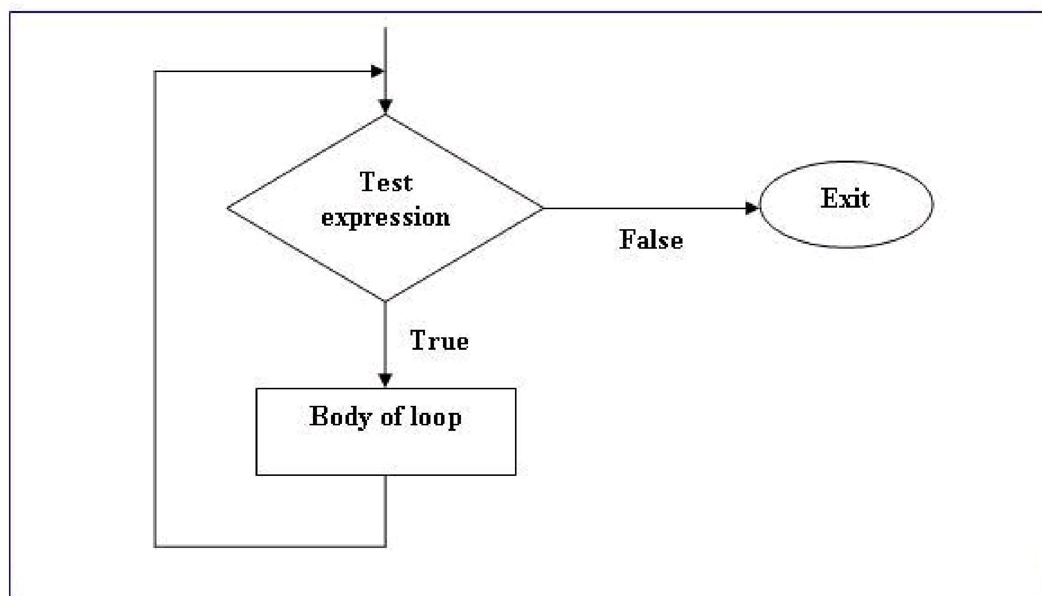


Figure 2 - Flow chart of the while loop

I told you before that the while loop is like the for loop, these are the similar aspects:

1. **You can use break statement and continue in both of them.**
2. **You can single or multi statements in the body of the loop in both of them, in the case of using multi statements you have to delimit them by braces.**
3. **The similar copy of for(;;) is while(true)**

Lesson 6 - Loops & Decisions (Part2)

Welcome to the sixth lesson in my course about MQL4.

I hope you enjoyed the previous lessons.

In the previous lesson, we have talked about the Loops.

And we have seen that the Loops are one of two ways we use to change the normal flow of the program execution -from top to bottom. The second way is the Decisions.

Decisions in a program cause a one-time jump to a different part of the program, depending on the value of an expression.

These are the kinds of decisions statements available in MQL4:

The if Statement

The if statement is the simplest decision statement, here's an example:

```
if( x < 100 )

Print("hi");
```

Here the if keyword has followed by parentheses, inside the parentheses the Test expression (x < 100),

when the result of test expression is true the body of the if will execute (Print("hi");) ,and if it is false, the control passes to the statement follows the if block.

Figure 1 shows the flow chart of the if statement:

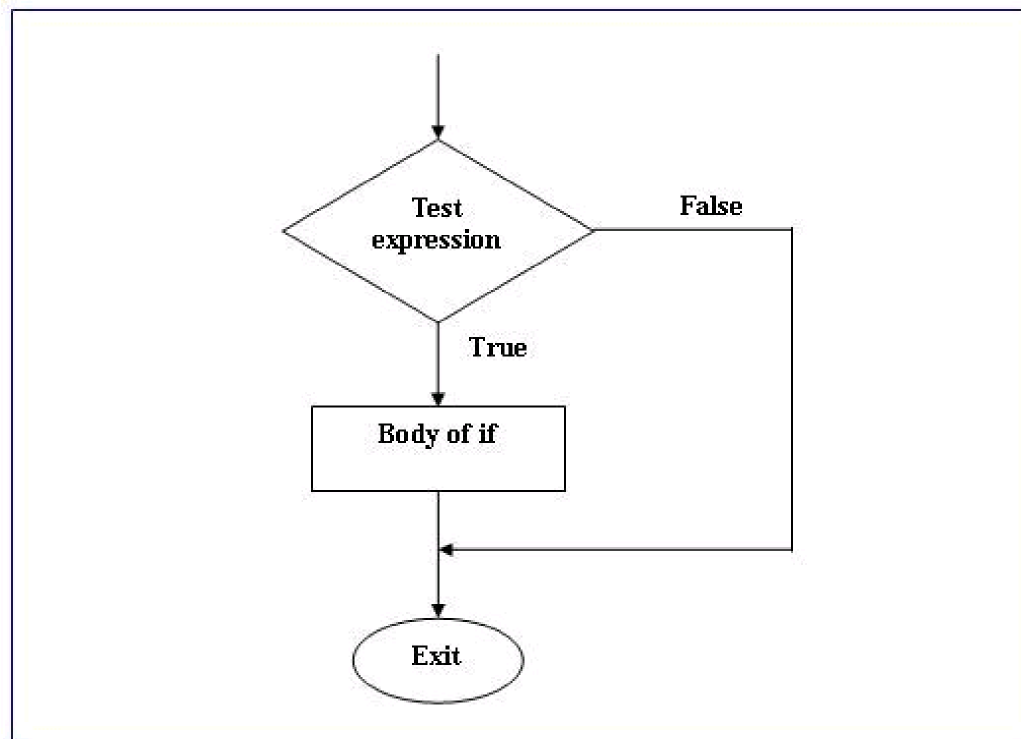


Figure 1 - Flow chart of the if statement

Multi Statements in the if Body:

Like the loops, the body of if can consist of more than statement delimited by braces.

For example:

```
if(current_price==stop_lose)
{
        Print("you have to close the order");
        PlaySound("warning.wav");
```

```
}
```

Notice the symbol == in the Test expression; it's one of the Relational Operators you have studied in the lesson 4, operations & expressions.

This is a source of a lot of errors, when you forget and use the assignment operator =.

Nesting:

The loops and decision structures can be basted inside one another; you can nest ifs inside loops, loops inside ifs, ifs inside ifs, and so on.

Here's an example:

```
for(int i=2 ; i<10 ; i++)
        if(i%2==0)
        {
                Print("It's not a prime nomber");
                PlaySound("warning.wav");
        }
```

In the previous example the if structure nested inside the for loop.

Notice: you will notice that there are no braces around the loop body, this is because the if statement and the statements inside its body, are considered to be a single statement.

The if...else Statement

The if statement let's you to do something if a condition is true, suppose we want to do another thing if it's false. That's the if...else statement comes in.

It consist of if statement followed by statement or a block of statements, then the else keyword followed by another statement or a block of statements.

Like this example:

```
if(current_price>stop_lose)

        Print("It’s too late to stop, please stop!");

else

        Print("you playing well today!");
```

If the test expression in the if statement is true, the program one message, if it isn’t true, it prints the other.

Figure 2 shows the flow chart of the if…else statement:

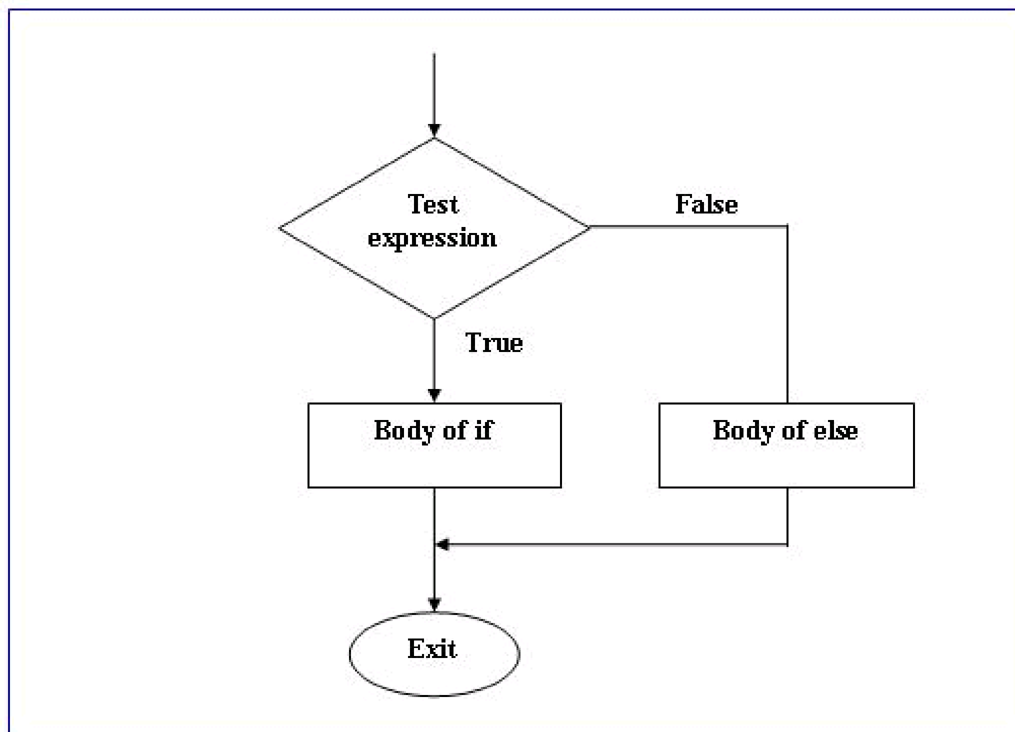


Figure 2 - Flow chart of the if..else statement

Nested if…else Statements

You can nest if…else statement in ifs statements, you can nest if…else statement in if…else statement, and so on.

Like this:

```
if(current_price>stop_lose)
        Print("It’s too late to stop, please stop!");
if(current_price==stop_lose)
        Print("It’s time to stop!");
else
        Print("you playing well today!");
```

There’s a potential problem in nested if…else statements, you can inadvertently match an else with the wrong if.

To solve this case you can do one of two things:

1- you can delimited the if…else pairs with braces like this:

```
if(current_price>stop_lose)
{
        Print("It’s too late to stop, please stop!");
if(current_price==stop_lose)
        Print("It’s time to stop!");
else
        Print("you playing well today!");
```

```
}
```

2- If you can't do the first solution (in the case of a lot of if…else statements or you are lazy to do it) take it as rule.

Match else with the nearest if. (Here it's the line if(current_price==stop_lose)).

The switch Statement

--

If you have a large decision tree, and all the decisions depend on the value of the same variable, you can use a switch statement here.

Here's an example:

```
switch(x)
  {
   case 'A':
     Print("CASE A");
     break;
   case 'B':
   case 'C':
     Print("CASE B or C");
          break;
   default:
     Print("NOT A, B or C");
     break;
  }
```

In the above example the switch keyword is followed by parentheses, inside the parentheses you'll find the

switch constant, this constant can be an integer, a character constant or a constant expression. The constant expression mustn't include variable for example:

case X+Y: is invalid switch constant.

How the above example works?

The switch statement matches the constant x with one of the cases constants.

In the case x=='A' the program will print "CASE A" and the break statement will take you the control out of the switch block.

In the cases x=='B' or x=='C', the program will print "CASE B or C". That's because there's no break statement after case 'B':.

In the case that x != any of the cases constants the switch statement will execute the default case and print "NOT A, B or C".

Figure 3 shows the flow chart of the switch statement

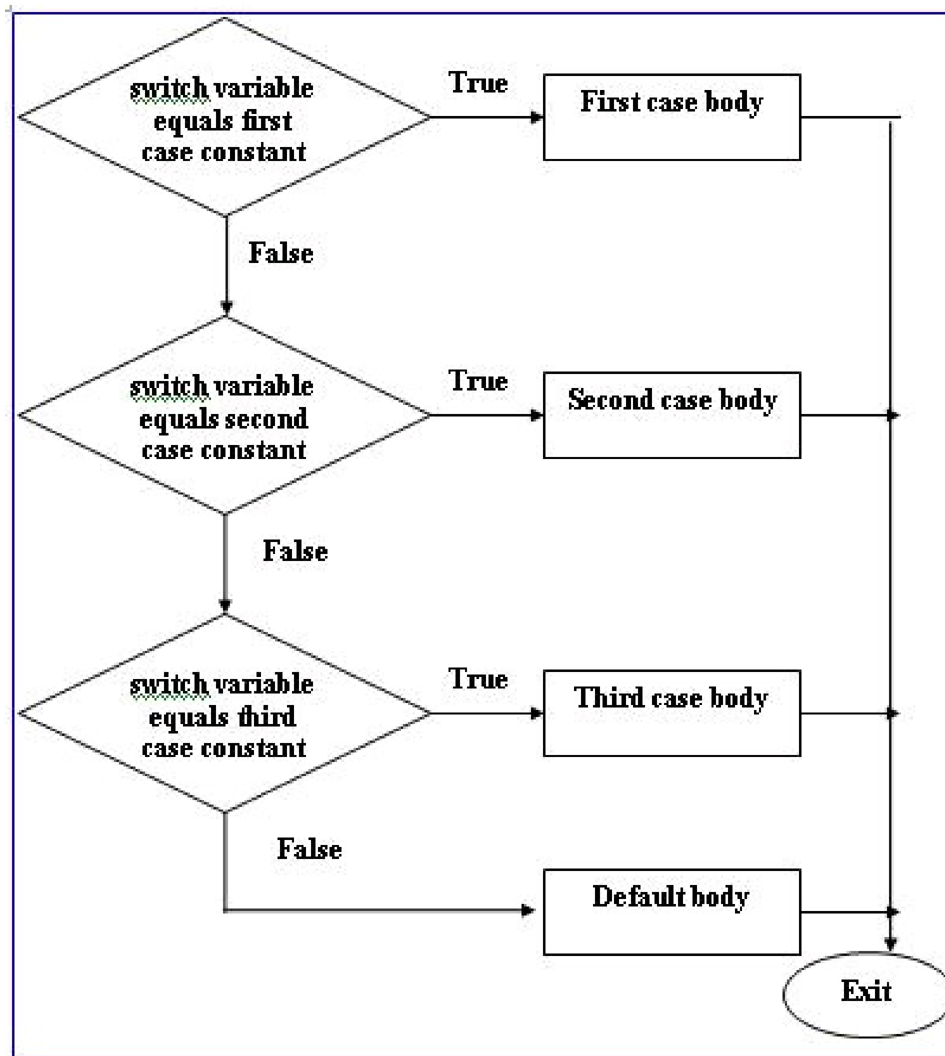


Figure 3 - Flow chart of the switch statement

Lesson 7 - Functions

Welcome to the world of MQL4 Functions.

The functions in any language take two phases:

Learning them which sometimes a boring thing.

Using them which always a lifeboat.

Let's start the seventh lesson.

What's the meaning of functions?

The function is very like the sausage machine, you input the meat and the spices and it outs the sausage.

The meat and the spices are the function parameters; the sausage is the function return value. The machine itself is the function body.

There's only one difference between the functions and your sausage machine, some of the functions will return nothing (nothing in MQL4 called void).

Let's take some examples:

```
double                          // type of the sausage – return value
my_func (double a, double b, double c) // function name and parameters list (meat & spices)
  {
   return (a*b + c);                // sausage outs - returned value
  }
```

As you see above, the function starts with the type of the returned value "double" followed by the function name which followed by parentheses.

Inside the parentheses you put the meat and spices, sorry, you put the parameters of the function.

Here we have put three parameters double a, double b, double c.

Then the function body starts and ends with braces. In our example the function body will produce the operation (a*b + c).

The return keyword is responsible about returning the final result.

Return keyword:

The return keyword terminate the function (like the break keyword does in the loop), and it gives the control to the function caller (we will know it soon).

The return keyword can include an expression inside its parentheses like the above example return (a*b + c); and this means to terminate the function and return the result of the expression.

And it can be without expression and its only job in this case is to terminate the function.

Notice: Not all the functions use the return keyword, especially if there's no return value. Like the next

example:

```
void                        // void mean there's no sausage – returned value.
my_func (string s) // function name and parameters list (meat & spices)
  {
     Print(s);
  }
```

The function above will not return value, but it will print the parameter s you provided. When the function has no return value you use "void" as the funciotn returns type.

These kinds of functions in some programming language called "Methods", but MQL4 calling them functions.

Function call:

We know very well now what the function is (I hope)? How to use the functions in your MQL4?

There's an extra steps after writing your function to use the function in you program.

This step is calling it (using it).

Assume you have a function which collects the summation of two integers.

This is the function:

```
int collect (int first_number, int second_number)
  {
     return(first_number+ second_number);
  }
```

You know how the previous function works, but you want to use it.

You use it like this:

```
int a = 10;

int b = 15;

int sum = collect(a,b);

Print (sum);
```

The example above will print 25 (is it a magic). But how did it know?

The magic line is int sum = collect(a,b); here you declared a variable (sum) to hold the function return value and gave the function its two parameters (a,b).

You basically called the function.

MQL4 when see your function name, it will take you parameters and go to the function and it will return –soon- with the result and place them in same line.

It's very like copying all the lines of the function instead of the place you called the function in, easy right?

Nesting functions inside function:

You can nest function (or more) inside the body of another function. That's because the caller line is treated like any normal statement (it's actually a statement).

For example:

We will use the collect function described above inside another new function which its job is printing the result of the collection:

```
void print_collection (int first_number, int second_number)
  {
     int sum = collect(first_number, second_number);
     Print(sum);
  }
```

Here we called the collect function inside the print_collection function body and printed the result. void means there's no return vale (do you still remember?).

MQL4 Special functions init(), deinit() and start():

In MQL4, every program begins with the function “init()” (initialize) and it occurs when you attach your program(Expert advisor or Custom indicator) to the MetaTrader charts or in the case you change the financial symbol or the chart periodicity. And its job is initializing the main variables of your program (you will know about the variables initialization in the next lesson).

When your program finishes its job or you close the chart window or change the financial symbol or the chart periodicity or shutdown MetaTrader terminal, the function "deinit()" (de-initialize) will occur.

The third function (which is the most important one) “start()” will occur every time new quotations are received , you spend 90 of your programming life inside this function.

We will know a lot about these functions in our real world lessons when we write our own Expert advisor and Custom Indictor.

Lesson 8 - Variables in MQL4

What are the variables mean?

As I told you the secret before, the variables are the names that refer to sections of memory into which data can be stored.

To help you think of this as a picture, imagine that memory is a series of different size boxes. The box size is memory storage area required in bytes.

- **In order to use a box to store data, the box must be given a name; this process is known as declaration.**
- **In the declaration process you use a word tell the computer what’s the kind and size of the box you want to use, this word known as keyword.**
- **It helps if you give a box a meaningful name that relates to the type of information which make it easier to find the data, this name is the variable constant.**
- **Data is placed into a box by assigning the data to the box.**
- **When we set the value of the box you have created in the same line you declared the variable; this process is known as initialization.**

When we create a variable we are telling the computer that we want him to assign a specified memory length (in bytes) to our variable, since storing a simple number, a letter or a large number is not going to occupy the same space in memory, so the computer will ask us what’s the kind of data and how much the length of the data? That is the Data type for.

For example if we said this line of code to the computer:

```
int  MyVaraible=0;
```

That’s mean we are asking the computer to set a block of 4 bytes length to our variable named “MyVaraiable”.

In the previous example we have used:

int ß Keyword

int ß Integer data type.

int ß Declaration

MyVaraible ß Variable's constant.

=0 ß Initialization

We will know more about variables in a coming lesson.

In MQL4, these are the kinds of Data types:

- **Integer (int)**
- **Boolean (bool)**
- **Character (char)**
- **String (string)**
- **Floating-point number (double)**
- **Color (color)**
- **Datetime (datetime)**

I've copied the previous few lines from the DATA TYPES lesson for you. To know what's the variable, now how do to declare the variables:

Declaration:

Declaring a variable means to introduce it to the world and specify its type. By using the keywords you have learned in the DATA TYPES lesson (int, double, char, bool, string, color and datetime) with the name you chose to the variable.

For example:

```
int  MyVaraible;
```

Here you declared a variable named MyVaraible which is an integer type. And before the declaration you can't use the MyVariable in your code. If you used it without declaration the MQL4 compiler will complain and will tell you something like this:'MyVaraible' - variable not defined. 1 error(s), 0 warning(s).

Initialization:

Initializing the variable means to assign a value to it, for example MyVaraible=0;   You can initialize the variable at the same line of the declaration like the example: int MyVaraible=0;

And you can declare the variable in one place and initialize it in another place like this:

```
int  MyVaraible;
…
…
MyVaraible=5;
```

But keep in your mind this fact: the declaration must be before the initialization.

Scopes of variables:

There are two scopes of the variables, Local and Global.

Scope means, which part of code will know about the variable and can use it.

Local variable means they are not seen to outside world where they had declared. For example the variables declared inside function are local to the function block of code, and the variables declared inside the loop or decisions block of code are local to those blocks and can be seen or used outside them.

For example:

```
double my_func (double a, double b, double c)
  {
   int d ;
   return (a*b + c);
  }
```

In the above example the variables a,b,c and d are local variables, which can be used only inside the function block of code (any thing beside the braces) and can't be used by outside code. So we can't write a line after the function above saying for example: d=10; because d is not seen to the next line of the function because it's outside it.

The second kind of the scopes is the Global variables, and they are the variables which had declared outside any block of code and can be seen from any part of your code.

For example:

```
int Global_Variable;
double my_func (double a, double b, double c)
  {
     return (a*b + c + Global_Variable);
  }
```

Here the variable Global_Variable declared outside the function (function level declaration) so, it can be seen by all the functions in you program.

The Global variables will automatically set to zero if you didn't initialize them.

Extern variables:

The keyword "extern" used to declare a special kind of variables; those kinds of variables are used to define input date of the program, which you can set them form the property of your Expert advisor or Custom indicator.

For example:

```
extern color Indicator_color = C'0x00,0x00,0xFF'; // blue

int init()

  {

   ...

  }
```

Here the variable Indicator_color had defined as an extern variable which you will see it the first time you attach your indicator (or EA) to the MetaTrader chart and which you can change it from the properties sheet windows. Look at Figure 1.

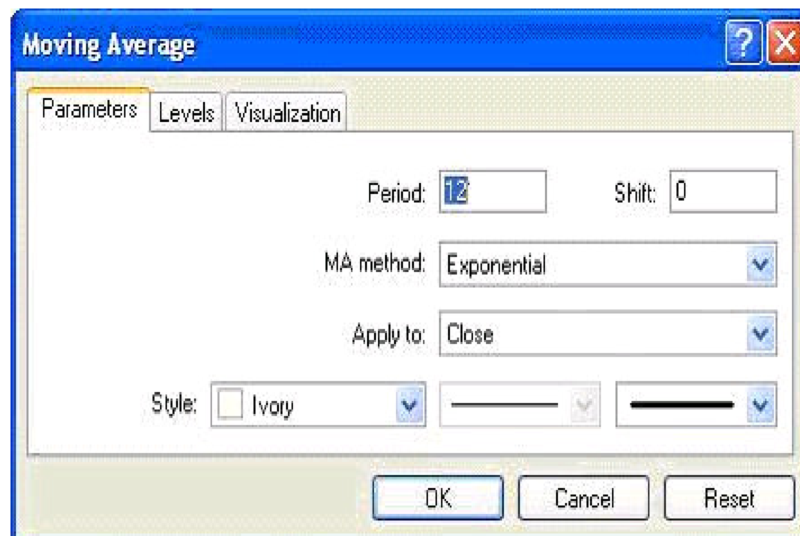


Figure 1: Property sheet of MA indicator

Here the variables Period, Shift, MA_method, Apply_to and Style are variables defined using the "extern" keyword so they appear in the property sheet.

Any variable you want the user of your program be able to change and set, make it extern variable.

Lesson 9 - Preprocessors

What are the Preprocessors mean?

Preprocessors are the instructions you give to the compiler to carry them out before starting (processing) your code.

For example if you used the preprocessor directive #include <win32.h> that's mean you telling the compiler to include the content of the file "win32.h" in the place you wrote the include keyword before processing your code.

In MQL4 there are four of preprocessors directives:

1- define directive:

define directive used to generate a constant.

The constant is very like the variable with only one different, you set its value only once and you can not change its value in your code like the variable.

For example:

```
#define my_constant      100
```

As you can notice in the above example there's no assignment symbol (=) but only space between the constant name (my_constant) and its value (100).

And you can notice too that the line didn't end with semi-colon but it ended with a carriage-return character (new line).

The name of constant obeys the same rules you had learnt about choosing the identifier names (lesson 2 SYNTAX), for example you can't start the constant name with a number or exceeds 31 characters.

The value of the content can be any type you want.

The compiler will replace each occurrence of constant name in your source code with the corresponding value.

So you can use the above constant in your code like that:

```
sum = constant1 * 10;
```

2- property directive:

There are predefined constants called “Controlling Compilation” included in the MQL4 language, which you can set them in your program.

They are the properties of your program which you can set them using the compiler directive “property” and the compiler will write them in the settings of your executable program (ex4 file).

For example:

```
#property link        "http://www.forex-tsd.com"
#property copyright   "Anyone wants to use"
```

This is the list of the MQL4 predefined constants:

Constant	Type	Description
link	string	a link to the company website
copyright	string	the company name
stacksize	int	stack size
indicator_chart_window	void	show the indicator in the chart window
indicator_separate_window	void	show the indicator in a separate window
indicator_buffers	int	the number of buffers for calculation, up to 8
indicator_minimum	int	the bottom border for the chart
indicator_maximum	int	the top border for the chart
indicator_colorN	color	the color for displaying line N, where N lies between 1 and 8
indicator_levelN	double	predefined level N for separate window custom indicator, where N lies between 1 and 8
show_confirm	void	before script run message box with confirmation appears
show_inputs	void	before script run its property sheet appears; disables show_confirm property

3- include directive:

When you asking the compiler to include a file name with the "include" directive, it's very like when you copy the entire file content and paste it in the place of the line you write include.

For example:

```
#include <win32.h>
```

In the above example you telling the compiler to open the file "win32.h" and reads all of its content and copy them in the same place of the include statement.

Note: in the above example you enclosed the file name with Angle brackets () and that's mean you telling the compiler to use the default directory (usually, terminal_directory\experts\include) to search for the file win32.h and don't search the current directory.

If the file you want to include located at the same path of your code, you have to use quotes instead of angle brackets like this:

```
#include "mylib.h"
```

In the both cases if the file can't be found you will get an error message.

You can use include at anywhere you want but it usually used at the beginning of the source code.

Tip: ***It's a good programming practice to write the frequently used code in a separate file and use include directive to put it in your code when you need (just an advice).***

4- import directive:

It's like include directive in the aspect of using outside file in your program.

But there are differences between them.

You use import only with MQL4 executables files (.ex4) or library files (.dll) to import their functions to your program.

For example:

```
#import "user32.dll"
   int MessageBoxA(int hWnd,string lpText,string lpCaption,
                   int uType);
   int MessageBoxExA(int hWnd,string lpText,string lpCaption,
                     int uType,int wLanguageId);
#import "melib.ex4"
#import "gdi32.dll"
   int      GetDC(int hWnd);
   int      ReleaseDC(int hWnd,int hDC);
#import
```

When you import functions from "ex4" file you haven't to declare their functions to be ready for use.

While importing the functions from a ".dll" file requires you to declare the functions you want to use like this:

```
int MessageBoxA(int hWnd,string lpText,string lpCaption,
                int uType);
```

And only the functions you has declared you can use in your code.

You must end the import directives with a blank import line #import (without parameters).

Lesson 12 - Your First Indicator (Part3)

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:

_

_ _ _ _ _ _ _ _ _ — — _ _ _ _ _ _ _ _ _ — _ _

_

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

Lesson 13 - Your First Expert Advisor (Part 1)

In the previous lesson we created our first indicator. Although this indicator wasn't useful for us as trader, but it was very useful for us as programmers.

The indicators –in general- are very important for the technical analysis of the market in trying to predict the future prices.

But with the indicators we observe the chart then use our hands to sell, buy and modify our orders manually. You have to set in front of your terminal, and keep your eyes widely open.

If you get tired, want to drink a cup of tea or even take a short vacation. You have to consider one of these solutions:

You may rent someone to observe the terminal for you and calling your mobile phone every five minutes to tell you what's going on. If this employee is an expert, he will cost you the pips you earn. And if he is novice one, he will cost you your capital.

The second solution is using a program to automate your trades.

That's what the Expert Advisor is for.

The Expert advisor is a program wrote in MQL4 (we are studying MQL4 huh?) uses your favorite indicators and trade methods to automate your orders.

It can buy, sell and modify the orders for you. It enables you to drink a cup of tea and save the salary you gave out to your employee or the bunch of flowers you bring to your assistant wife.

Today we are going to create our first expert advisor so let's go.

First two steps:

Step1:

If you didn't open your MetaEditor yet, I think it's the time to run it.

From the MetaEditor File menu click New (you can use CTRL+N hotkey or click the New icon in the standard toolbar). That will pop up the new program wizard which you have seen when you created your first indicator (Figure 1).

This time we will choose the first option "Expert Advisor program" then click Next button.

Figure 1 – the first step wizard

Step2:

When you clicked Next, you have got the general properties wizard (Figure 2).

This wizard enables you to set the properties of your expert advisor and to set the external variables you will use in your expert advisor.

In this step you can set these properties:

1- Name of your expert advisor, in our sample we will use My_First_EA.

2- Author name of the program, type your name (I typed mine in the sample).

3- Link to your web site.

4- External variables list:

This is the list of the external variables you allow the user of your expert advisor to change them from the Expert properties window.

To add a new variable you click the Add button, clicking it will add a new record to the external variables list. Every record has three fields:

Name: double click this field to set the name (identifier) of the variable.

Type: double click this field to set the data type of the variable.

Initial value: double click this field to give your variable initialization value.

This field is *optional* which means you can leave it without setting

In our sample we have added three variables:

Varaible _ Type _ initial value

TakeProfit _ double _ 350

Lots _ double _ 0.1

TrailingStop _ double _ 35

Figure 2 – the second step wizard

Now click the Finish button to close the wizard, and MetaEditor will bring to you the code created by the wizard and saves the file My_First_EA.mq4 in the MetaTrader 4 \experts path.

Note: you have to put the expert advisors in MetaTrader 4\experts path and the indicators in MetaTrader 4\experts\indicators path, otherwise they will not work.

This is the code we have got from the wizard:

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

As you see above, the code generated by the wizard is a template for you to add your code without bothering you by typing the main functions from scratch.

Now let's add our own code:

```
//+------------------------------------------------------------------+
//| 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_dirction = 0;
if(line1>line2)current_dirction = 1; //up
if(line1<line2)current_dirction = 2; //down
if(current_dirction != last_direction) //changed
{
last_direction = current_dirction;
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);
}
//+------------------------------------------------------------------+
```

Note: don't copy and paste the code above because it warped and will not work for you, use the code provided with lesson in www.forex-tsd.com .

Scared?

Don't be scared of the 160 lines you have seen above, we will know everything about this code line by line, I promise it's an easy task.

Test the Expert Advisor:

Before studying our expert advisor code we have to be check is it profitable one or not.

Note: Our expert advisor will work with EURUSD pairs in 4 Hours timeframe.

So compile the expert advisor by pressing F5 and load it in MetaTrader.

You can test your expert advisor using two methods:

1- Live trading

In live trading testing the results are more accurate but you have to spend days (and maybe months) to know is the expert advisor profitable or not.

You have to enable the expert advisor to automate your trades.

To enable it click Tools _ Option menu (or use CTRL+O hotkey), that's will bring the Options windows (figure 3), click Expert Advisors tab and enable these options:

Enable Expert Advisors

Allow live trading

And click Ok button

Figure 3 – Enabling expert advisor auto trade

You will see the Smile symbol beside the expert advisor name which means your expert advisor is working and ready to trade for you (Figure 4).

Figure 4 – Expert advisor is enabled

2- Strategy Tester:

The second method of testing your expert advisor which is less accurate but will not take time is the Strategy tester. We will know everything about Strategy tester later, let's now bring its window by pressing F6 (Figure 5).

When you get the window enter these options:

Symbol: EURUSD.

Period: H4 (4 Hours).

Model: Open price only.

Figure 5 – Strategy tester window

Now click Start button to start testing the expert advisor.

You will see a progress bar and the two tabs (Settings and Journal) became five tabs.

We interested in Report tab (Figure 6); click it to see your profit.

We have a lot to say and to do in the next lesson; I hope you are ready for the challenge.

I welcome very much the questions and the suggestions._

See you

Coders' Guru

Lesson 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

Lesson 15 - Your First Expert Advisor (Part 3)

In the previous two parts of this lesson, we have introduced our expert advisor and knew the idea behind it.

And in the *Appendix 2* we have studied the *Trading Functions* which we will use some of them today in our expert advisor.

Today we will continue cracking the remaining code of the expert advisor.

I hope you have cleared your mind for our discovering mission.

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);
}
//+------------------------------------------------------------------+
int cnt, ticket, total;
```

In this line we declared three *integer* type variables.

We used one line to declare the three of them because they are the same type (You can't use a single line declaration with dissimilar data types).

Note: To declare multi variables in a single line you start the line with the declaration keyword which indicates the type of the variables then separate the variables identifiers (names) with comma.

You can divide the above line into three lines like that

int cnt;

int ticket;

int total;

We will use the *cnt* variable as a counter in our "opened orders checking loop".

We will use the *ticket* variable to hold the ticket number returned by *OrderSend* function.

And we will use the *total* variable to hold the number of already opened orders.

```
double shortEma, longEma;
```

Again, we used a single line to declare two *double* variables.

We will use these variables to hold the value of short EMA and long EMA.

As you remember (I hope) from the previous part we uses the *Crossing* of short and long EMAs as buying and selling conditions and as closing conditions too.

```
if(Bars<100)
{
Print("bars less than 100");
return(0);
```

}

We want to work with a normal chart and we assume that the normal chart must have more than 100 bars. That's because the insufficient numbers of bars will not enable or EMA indicators to work right.

We have got the number of bars on the chart using the *Bars* function and checked this number to find is it less than 100 or not. If it's less than 100 we will do two things:

We will tell the user what's the wrong by writing this message to the Experts log *"bars less than 100"* (*Figure 1*).

Then we will terminate the *start* function by the line *retrun(0);*

That's the way we refuse to work with less than 100 bars on the chart.

Figure 1 – Experts log

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

We don't want too to work with insufficient value of *TakeProfit*.

The *TakeProfit* variable is an external variable and that's mean the user can change its default value from the expert advisor properties windows.

We want to protect the user of our expert advisor from his bad choices.

We assume that any value less than 10 for the *TakeProfi variable* will be bad choice and for that we have checked the *TakeProfit* value the user set to find is it less than 10 or not.

If it's less than 10 we will inform the user what's the wrong by printing him the message *"TakeProfit less than 10"*, and we will terminate the *start* function by *return(0)* statement.

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

Well, everything is OK, the bars on the chart were more than 100 bars and the value of *TakeProfit* the user supplied was more than 10.

We want now to calculate the short and long EMAs of the current bar.

We use the MQL4 built-in technical indicator function iMA which calculates the moving average indicator.

I have to pause here for a couple of minutes to tell you more details about iMA function.

iMA:

Syntax:

double iMA(string symbol, int timeframe, int period, int ma_shift, int ma_method, int applied_price, int shift)

Description:

The *iMA* function calculates the moving average indicator and returns its value (double data type).

Note: A moving average is an average price of a certain currency over a certain time interval (in days, hours, minutes etc) during an observation period divided by these time intervals.

Parameters:

This function takes 7 parameters:

string symbol:

The symbol name of the currency pair you trading (Ex: EURUSD and USDJPY).

If you want to use the current symbol use NULL as a parameter.

int timeframe:

The time frame you want to use for the moving average calculation.

You can use one of these time frame values:

Constant Value Description

PERIOD_M1 1 1 minute.

PERIOD_M5 5 5 minutes.

PERIOD_M15 15 15 minutes.

PERIOD_M30 30 30 minutes.

PERIOD_H1 60 1 hour.

PERIOD_H4 240 4 hour.

PERIOD_D1 1440 Daily.

PERIOD_W1 10080 Weekly.

PERIOD_MN1 43200 Monthly.

0 (zero) 0 Time frame used on the chart.

If you want to use the current timeframe, use 0 as a parameter.

Note: You can use the integer representation of the period or its constant name.

For example, the line:

iMA(NULL, PERIOD_H4,8,0,MODE_EMA,PRICE_CLOSE,0);

is equal to

iMA(NULL,240,8,0,MODE_EMA,PRICE_CLOSE,0);

But it's recommended to use the constant name to make your code clearer.

int period:

The number of days you want to use for the moving average calculation.

int ma_shift:

The number of the bars you want to shift the moving average line relative to beginning of the chart:

0 means no shift (*Figure 2*)

A positive value shifts the line to right (*Figure 3*)

A negative value shifts the line to left (*Figure 4*)

int ma_method:

The moving average method you want to use for the moving average calculation,

It can be one of these values:

Constant Value Description

MODE_SMA 0 Simple moving average.

MODE_EMA 1 Exponential moving average.

MODE_SMMA 2 Smoothed moving average.

MODE_LWMA 3 Linear weighted moving average.

int applied_price:

The price you want to use for the moving average calculation,

It can be one of these values:

Constant Value Description

PRICE_CLOSE 0 Close price.

PRICE_OPEN 1 Open price.

PRICE_HIGH 2 High price.

PRICE_LOW 3 Low price.

PRICE_MEDIAN 4 Median price, (high+low)/2.

PRICE_TYPICAL 5 Typical price, (high+low+close)/3.

PRICE_WEIGHTED 6 Weighted close price, (high+low+close+close)/4.

int shift:

The number of bars (relative to the current bar) you want to use for the moving average calculation. Use 0 for the current bar.

Figure 2 : ma_ shift = 0

Figure 3: ma_shift = 10

Figure 4: ma_shift = -10

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

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

Now you know what the above lines means.

We assigned to the variable *shortEma* the value of:

8 days closing price based exponential moving average of the current bar.

Which we can briefly call it 8EMA

And assigned to the variable *longEma* the value of:

13 days closing price based exponential moving average of the current bar.

Which we can briefly call it 13EMA

int isCrossed = Crossed (shortEma,longEma);

Note: 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.

Here we declared an integer variable *isCrossed* to hold the return value of the function *Corssed*. We will use this value to open and close orders.

total = OrdersTotal();

if(total < 1)

```
{
…….
}
```

We assigned the *OrdersTotal* return value to the variable *total.*

Note: The *OrdersTotal* function returns the number of opened and pending orders. If this number is 0 that means there are no orders (market or pending ones) has been opened.

Please review appendix 2

Then we checked this number (total) to find if there was already opened orders or not. The *if block* of code will work only if the total value is lesser than 1 which means there's no already opened order.

```
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);
}
```

In the case the *shortEma* crossed the *longEma* and the *shortEma* is above the *longEma* we will buy now.

We are using the *OrderSend* function to open a buy order.

Note: The *OrderSend* function used to open a sell/buy order or to set a pending order. It returns the ticket number of the order if succeeded and -1 in failure.

Syntax:

```
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=CLR_NONE)
```

Please review appendix 2

These are the parameters we used with our *OrderSend* function:

symbol:

We used the *Sybmol* function to get the symbol name of the current currency and passed it to the *OrderSend* function.

cmd:

We used *OP_BUY* because we want to open a Buy position.

volume:

We used the *Lots* value that the use has been supplied.

price:

We used the *Ask* function to get the current ask price and passed it to the OrderSend function.

slippage:

We used 3 for the *slippage* value.

stoploss:

We used 0 for *stoploss* which mean there's no *stoploss* for that order.

takeprofit:

We multiplied the *TakeProfit* value the user has been supplied by the return value of the *Point* function and added the result to the *Ask* price.

Note: *Point* function returns the point size of the current currency symbol.

For example: if you trade EURUSD the point value = .0001 and if you trade EURJPY the point value should be .01

So, you have to convert your *stoploss* and *takeprofit* values to points before using them with *OrderSend* or *OrderModify* functions.

comment:

We used *"My EA"* string for the comment

magic:

We used the number *12345* for the order *magic* number.

expiration:

We didn't set an *expiration* date to our order, so we used 0.

arrow_color:

We set the color of opening arrow to *Green* (Because we like the money and the money is green)

The *OrderSend* function will return the ticket number of the order if succeeded, so we check it with this line:

```
if(ticket>0)
```

We used the *OrderSelect* function to select the order by its ticket number that's before we used the *OrderOpenPrice* function which returns the open price of the selected order.

Everything is OK, the *OrderSend* returned a proper ticket number (greater than 0) and the *OrderSelect* successfully selected the order. It's the good time to tell the user this good new by printing to him the message *"BUY order opened : "* plus the opened price of the order.

Note: For more details about *OrderSelect* and *OrderOpenPrice*, please review appendix 2

Else, if the *OrderSend* function returned -1 which means there was an error opening the order, we have to tell the user this bad news by printing him the message: *"Error opening BUY order : "*, plus the error number returned by the function *GetLastError*.

And in this case we have to terminate the start function by using *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);
}
```

Here, the opposite scenario, the *shortEma* crossed the *longEma* and the *shortEma* is below the *longEma* we will sell now.

We use the *OrderSend* function to open a Sell order. Can you guess what the differences

between the Buy parameters and Sell parameters in *OrderSend* are? Right! You deserve 100 pips as a gift.

These parameters haven't been changed:

symbol is the same.

volume is the same.

slippage is the same.

stoploss is the same.

comment is the same.

magic is the same.

expiration is the same.

These parameters have been changed (that's why you've got the gift):

cmd:

We used *OP_SELL* because we want to open a Sell position.

price:

We used the *Bid* function to get the current bid price and passed it to the *OrderSend* function.

takeprofit:

We multiplied the *TakeProfit* value the user has been supplied by the return value of the *Point* function and subtracted the result from the *Bid* price.

arrow_color:

We set the color of opening arrow to *Red* (Because we like the money and the money is green but we want a different color for selling position).

We can briefly repeat what will happen after calling the *OrderSend* with the above parameters:

The *OrderSend* returns the ticket number if succeeded.

We check this number to find is it greater than 0 which means no errors.

We used the *OrderSelect* to select the order before using *OrderOpenPrice*.

Everything is OK, we tell the user this good news by printing to him the message *"Sell order opened : "* plus the opened price of the order.

Else, if the *OrderSend* function returned -1 we tell the user this bad news by printing him the message: *"Error opening SELL order : "*, plus the error number and terminate the *start*

function by using *return(0).*

- Wait a minute! (You said).

- I've noticed that there's a line after the last *if* block of code you have just explained above.

- Where? (I'm saying).

- Here it's:

return(0);

Yes! Great but I have no pips to gift you this time.

Look carefully to these blocks (braces):

```
if(total < 1)

{

if(isCrossed == 1)

{

.....

}

if(isCrossed == 2)

{

.....

}

return(0); _ Right!

}
```

If the *shorEma* crossed the *longEma* upward we open Buy order.

If the *shorEma* crossed the *longEma* downward we open Sell order.

What if they haven't been crossed yet? That's the job of this line.

We terminate the start function if there was no crossing.

(In the case that *isCrossed* not equal 1 or 2).

Now we are ready to open Buy and Sell positions.

In the coming part we will crack the remaining of the code and will discuss in details the

MetaTrader Strategy Tester.

I hope you enjoyed the lesson.

I welcome very much your questions and suggestions.

Coders' Guru

Account Information functions

Hi folks,

Regardless the type of your account, demo account or a real one; it's very important to have a set of functions to access this account information, for example the current account balance, the current account margin value and the current account total profit.

The account information functions are very useful for applying Money Management techniques by calculating the state of your account balance and margin etc.

Let's study these functions in details.

AccountBalance

Syntax:

```
double AccountBalance()
```

Description:

The *AccountBalance* function returns the balance amount (how much money) of the current account. The calculation of the balance amount is the calculation of the starting capital + profit or - loss (of closed trades). The profit are loss are calculated only for the closed trades, so the floating profit/loss is not calculated within the balance!

Note: All of the Account Information function don't take parameters, see the examples!

Example:

```
Print("Account balance = ", AccountBalance());
```

AccountCredit

Syntax:

```
double AccountCredit()
```

Description:

The *AccountCredit* function returns the credit value of the current account

Example:

```
Print("Account number ", AccountCredit());
```

AccountCompany

Syntax:

```
string AccountCompany()
```

Description:

The *AccountCompany* function returns the brokerage company name you opened your current account with, the same as the name you find in the About dialog which you can open from Help menu.

Example:

```
Print("Account company name ", AccountCompany());
```

AccountCurrency

Syntax:

```
string AccountCurrency()
```

Description:

The AccountCurrency function returns the name of the currency you used to open your account with the brokerage company, for example USD if you paid your account in US dollars.

Example:

```
Print("account currency is ", AccountCurrency());
```

AccountEquity

Syntax:

```
double AccountEquity()
```

Description:

The *AccountEquity* function returns the equity amount of the current account. The calculation of the balance equity is the calculation of the starting capital + profit or - losses (of all trades). The profit are loss are calculated for the opened and the closed trades.

Example:

```
Print("Account equity = ",AccountEquity());
```

Conversion functions

Hi folks,

Today we are going to talk about a very important set of MQL4 functions; the Conversion functions.

Why should I use Conversion functions?

In many of situations you have a variable in a specific format and you want to use it in another format.

For example: When we use the CurTime() function which returns the current server time; it returns this time in datetime format.
It will not problem for us if we are going to use this time as a datatime format for example adding to 2 hours to it or substracting it from the local time.

But if we want to display this time to the user using the Alert() function we will face a problem. the Alert() can't convert automatically from datetime data type to string data type which we want the user to see. In this case we have to use the conversion function TimeToStr().

Implicit conversions:

Beside the conversions functions we are going to study I have to mention that MQL4 will make implicit conversions from a data type to other data type when you assign a wrong value for this data type.

For example:

```
string var1 = 100;
Alert(var1);
```

In the above code you have assigned an integer to a string variable. MQL4 will convert the number 100 from integer to string.
If you don't sure of that add this line:

```
Alert(var1+10);
```

What do you think you'll get? No, not 110 but you will get 10010 (Figure 1). That's because MQL4 has converted the 100 to string and 10 to string then added them to each others as strings.

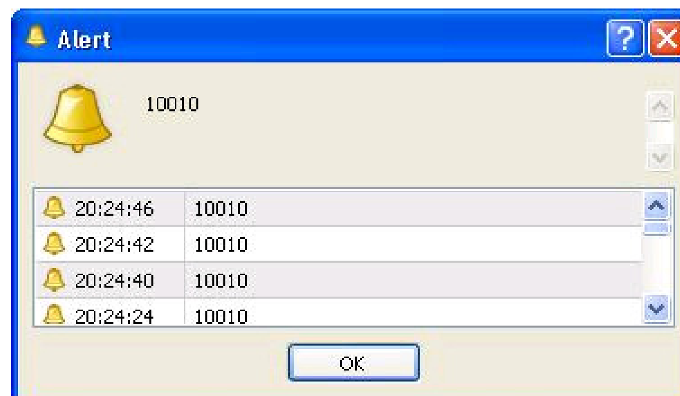


Figure 1

Let's study the conversion function available in MQL4!

CharToStr:

Syntax:

```
string CharToStr( int char_code)
```

Description:

The *CharToStr* function converts from Char type to string type; it converts the ASCII char code passed to it to string.

Parameters:

This function takes only one parameter:

int char_code

The ASCII char code of the character you want to convert it to string.

Example:

```
for (int cnt = 1 ; cnt < 255 ; cnt++)
{
Print("ASCII code: " + cnt + ": " + CharToStr(cnt));
}
```

DoubleToStr:

Syntax:

```
string DoubleToStr( double value, int digits)
```

Description:

The *DoubleToStr* function converts from double data type to string type; it converts the double value passed to the function to string with the number of digits passed to the function.

Parameters:

This function takes two parameters:

double value

The double value you want to convert it to string.

int digits

The number of digits you want the function to use in converting the double value to string. it can be ranged from 0 (no digits) to 8 (8 digits).

Example:

```
string value=DoubleToStr(1.2345678, 4);
// value is 1.2346
```

NormalizeDouble:

Syntax:

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

Description:

The *NormalizeDouble* function rounds the double value passed to it to the number of digits passed. It's like DoubleToStr function but it returns double value instead of string.

Parameters:

This function takes two parameters:

double value

The double value you want to round it.

int digits

The number of digits you want the function to use in rounding the double value. it can be ranged from 0 (no digits) to 8 (8 digits).

Example:

```
double value=1.2345678;
Print(NormalizeDouble(value,5));
// output: 1.2346
```

StrToDouble:

Syntax:

```
double StrToDouble( string value)
```

Description:

The *StrToDouble* function converts from string data type to double type; it converts the string value passed to the function to a double.

Parameters:

This function takes only one parameter:

string value

The string value you want convert it to double value.

Example:

```
double value=StrToDouble("1.2345678");
Print(value);
// output: 1.2346
```

StrToInteger:

Syntax:

```
int  StrToInteger( string value)
```

Description:

The *StrToInteger* function converts from string data type to integer type; it converts the string value passed to the function to an integer.

Parameters:

This function takes only one parameter:

string value

The string value you want convert it to integer value.

Example:

```
double value=StrToInteger("1999");
Print(value);
```

StrToTime:

Syntax:

```
datetime StrToTime( string value)
```

Description:

The *StrToTime* function converts from string data type to datetime data type; the string value passed to the function must be in the format: "yyyy.mm.dd hh:mi"

Parameters:

This function takes only one parameter:

string value

The string value you want convert it to datatime data type. This string have in one of these formats:
"yyyy.mm.dd hh:mi"
"hh:mi"
"yyyy.mm.dd"

Example:

```
datetime var1;
var1=StrToTime("2003.8.12 17:35");
var1=StrToTime("17:35"); // returns with current date
var1=StrToTime("2003.8.12"); // returns with midnight time "00:00"
```

TimeToStr:

Syntax:

```
string TimeToStr( datetime value, int mode=TIME_DATE|TIME_MINUTES)
```

Description:

The *TimeToStr* function converts from datetime data type to string data type; the return string value will be in the format "yyyy.mm.dd hh:mi".

Parameters:

This function takes two parameters:

datetime value

The datatime value you want to convert it to string. It starts from 00:00 January 1, 1970.

int mode

Optional parameter determine the mode of conversion; what the string format the function will return.
It can be one or combination of these modes:
TIME_DATE the result will be in the format "yyyy.mm.dd",
TIME_MINUTES the result will be in the format"hh:mi",
TIME_SECONDS the result will be in the format "hh:mi:ss".

The default value is TIME_DATE|TIME_MINUTES which means the result will be in the format "yyyy.mm.dd hh:mi".

Example:

```
strign var1=TimeToStr(CurTime(),TIME_DATE|TIME_SECONDS);
```

Date and Time Functions

Hi folks,

We are going to study today a very important set of MQL4 functions; the Date and Time Functions, which are the functions that returns the local (your computer) and server time.

As an example to show the importance of these function you can manage the time your expert advisor will enter the market and exist from the market. Another example with the aid of these functions you can know how long a position has been opened by subtracting the order opened time (returned by the function OrderOpenTime) from the current time (returned by he function CurTime which we are going to study it later).
There are a lot of useful usages of the date and time function that can't be listed here.

Before we dig into studying the date and time function we have to review the datetime data type because it's the cornerstone of understanding how the date and time is calculating in MQL4.

datatime data type:

datetime data type is a special MQL4 data type, which holds a date and time data. You set the datetime variable by using the keyword (D) followed by two signal quotations ('). Between the two signal quotations you write a character line consisting of 6 parts for value of year, month, date, hour, minutes, and seconds. datetime constant can vary from Jan 1, 1970 to Dec 31, 2037.

For example:

```
D'2004.01.01 00:00' // New Year
D'1980.07.19 12:30:27'
D'19.07.1980 12:30:27'
D'19.07.1980 12' //equal to D'1980.07.19 12:00:00'
D'01.01.2004' //equal to D'01.01.2004 00:00:00'
```

We use the keyword datetime to create a datetime variable.

For example:

```
datetime dtMyBirthDay= D'1972.10.19 12:00:00';
datetime dt1= D'2005.10.22 04:30:00';
```

Now, let's give the date and time functions a study trip:

CurTime:

Syntax:

```
datetime CurTime( )
```

Description:

The *CurTime* function returns the current server time, it's not always the exact current server time but it's the last know server time that the terminal has been retrieved it from the server with the last price quotation.

The return value of the CurTime function is a datetime data type and it's the number of seconds elapsed from 00:00 January 1, 1970.

What if you used this function (and all the date & time functions) in the testing mode - Back Testing?
In the testing mode the server time will be modeled by the tester.

Parameters:

This function takes no parameters and return datetime value.

Example:

```
if(CurTime()-OrderOpenTime()<360) return(0);
```

Day:

Syntax:

```
int Day( )
```

Description:

The *Day* function returns the current day of the month (1, 2, 3, 4, ... 31) of the last known server time.
The day is modeled in the testing mode.

Parameters:

This function takes no parameters and return integer value.

Example:

```
if(Day()<5) return(0);
```

DayOfWeek:

Syntax:

```
int DayOfWeek( )
```

Description:

The *DayOfWeek* function returns the current day of the week of the last know server time:

0 = Sunday
1 = Monday
2 = Tuesday
3 = Wednesday
4 = Thursday
5 = Friday
6 = Saturday

The day of week is modeled in the testing mode.

Parameters:

This function takes no parameters and return integer value.

Example:

```
// does not work on holidays.
if(DayOfWeek()==0 || DayOfWeek()==6) return(0);
```

DayOfYear:

Syntax:

```
int DayOfYear( )
```

Description:

The *DayOfYear* function returns the current day of the year (1, 2, 3, ... 365 (366)) of the last know server time.
The day of year is modeled in the testing mode.

Parameters:

This function takes no parameters and return integer value.

Example:

```
if(DayOfYear()==245) return(true);
```

Hour:

Syntax:

```
int Hour( )
```

Description:

The *Hour* function returns the current hour (0, 1, 2, ... 23) of the last know server time. The hour is modeled in the testing mode.

Note: The hour returned by the Hour function is the hour of the moment the program start and will not change during the execution of the program.

Parameters:

This function takes no parameters and return integer value.

Example:

```
if(Hour()>=12 || Hour()<17) return(true);
```

LocalTime:

Syntax:

```
datetime LocalTime( )
```

Description:

The *LocalTime* function returns the current local computer time. The return value of the LocalTime function is a datetime data type and it's the number of seconds elapsed from 00:00 January 1, 1970. The local time is modeled in the testing mode as well as the server time.

Parameters:

This function takes no parameters and return datetime value.

Example:

```
if(LocalTime()-OrderOpenTime()<360) return(0);
```

Minute:

Syntax:

```
int Minute( )
```

Description:

The *Minute* function returns the current minute (0, 1, 2, ... 59) of the last know server time. Like the Hour function the minute returned by the Minute function is the minute of the moment the program start and will not change during the execution of the program. The minute is modeled in the testing mode.

Parameters:

This function takes no parameters and return integer value.

Example:

```
if(Minute()<=15) return("first quarter");
```

Month:

Syntax:

```
int Month( )
```

Description:

**The *Month* function returns the current month (0, 1, 2, 3, ... 12) of the last know server time.
The month is modeled in the testing mode.**

Parameters:

This function takes no parameters and return integer value.

Example:

```
if(Month()<=5) return("the first half year");
```

Seconds:

Syntax:

```
int Seconds( )
```

Description:

**The *Seconds* function returns the current seconds (0, 1, 2, ... 59) of the minute of the last know server time. Like the Hour and the Minute functions the seconds returned by the Seconds function is the second of the moment the program start and will not change during the execution of the program.
The minute is modeled in the testing mode.**

Parameters:

This function takes no parameters and return integer value.

Example:

```
if(Seconds()<=15) return(0);
```

Year:

Syntax:

```
int Year( )
```

Description:

The *Year* function returns the current year of the last know server time.
The year is modeled in the testing mode.

Parameters:

This function takes no parameters and return integer value.

Example:

```
// return if the date is within the range from 1 Jan. to 30 Apr., 2006.
if(Year()==2006 && Month()<5) return(0);
```

TimeDay:

Syntax:

```
int TimeDay(datetime date)
```

Description:

The *TimeDay* function returns the day of the month (1, 2, 3, ... 31) of the given date. It extracts the day of the month from the given date

Parameters:

datetime date:

The date (datetime data type) you want to extract the day of the month from.

Example:

```
int day=TimeDay(D'2003.12.31');  // day is 31
```

TimeDayOfWeek:

Syntax:

```
int TimeDayOfWeek(datetime date)
```

Description:

The *TimeDayOfWeek* function returns the day of the week (0,1, 2, ... 6) of the given date. It extracts the day of the week from the given date

Parameters:

datetime date:

The date you want to extract the day of the week from.

Example:

```
int weekday=TimeDayOfWeek(D'2004.11.2'); // day is 2 - Tuesday
```

TimeDayOfYear:

Syntax:

```
int TimeDayOfYear(datetime date)
```

Description:

The *TimeDayOfYear* function returns the day of the year (1, 2, 3, ... 365 (366)) of the given date. It extracts the day of the year from the given date

Parameters:

datetime date:

The date you want to extract the day of the year from.

Example:

```
int day=TimeDayOfYear(CurTime());
```

TimeHour:

Syntax:

```
int TimeHour(datetime date)
```

Description:

The *TimeHour* function returns the hour of the day (0,1, 2, ... 23) of the given date. It extracts the hours of the day from the given date

Parameters:

datetime date:

The date you want to extract the hour of the day from.

Example:

```
int h=TimeHour(CurTime());
```

TimeMinute:

Syntax:

```
int TimeMinute(datetime date)
```

Description:

The *TimeMinute* function returns the minute of the hour (0,1, 2, ... 59) of the given date. It extracts the minute of the hour from the given date

Parameters:

datetime date:

The date you want to extract the minute of the hour from.

Example:

```
int m=TimeMinute(CurTime());
```

TimeMonth:

Syntax:

```
int TimeMonth(datetime date)
```

Description:

The *TimeMonth* function returns the month of the year (1, 2, 3, ... 12) of the given date. It extracts the

month of the year from the given date

Parameters:

datetime date:

The date you want to extract the month of the year from.

Example:

```
int m=TimeMonth(CurTime());
```

TimeSeconds:

Syntax:

```
int TimeSeconds(datetime date)
```

Description:

The *TimeSeconds* function returns the seconds of the minute (0,1, 2, ... 59) of the given date. It extracts the seconds of the minute from the given date

Parameters:

datetime date:

The date you want to extract the seconds of the minute from.

Example:

```
if(Seconds()<=15) return(0);
```

TimeYear:

Syntax:

```
int TimeYear(datetime date)
```

Description:

The *TimeYear* function returns the year of the given date. It extracts the year from the given date

Parameters:

datetime date:

The date you want to extract the year from.

Example:

```
// return if the date is within the range from 1 Jan. to 30 Apr., 2006.
if(Year()==2006 && Month()<5) return(0);
```

Hope you enjoyed the lesson!

Coders Guru

Examples: Money Management

Hi folks,

One of the most important keys (if not the most important one) of success trading is the Money Management technique you apply to your trading operations.

The Money Management is how to handle the money you put in one or more trades, where you put more money in the good trades and put less money in the bad trades. And at the same time how to handle the capital you using in your trading account where you spend more when you have more balance and spend less when you have less balance.

The most of traders think first in when to buy/sell but the successful ones think first how manage the risk before buying/selling
The Money Management briefly is the art of handling the size of the trade

Because we are in the development section we are taking another approach of talking about the Money Management, we are going to talk about a MQL4 code which enable us to apply our Money Managements and Risk controller techniques in the case of writing an expert advisor.

We have a simple here in MQL4.

```
//+------------------------------------------------------------------+
//|                                                   EMA_CROSS_2.mq4 |
//|                                                       Coders Guru |
//|                                            http://www.forex-tsd.com |
//+------------------------------------------------------------------+


#property copyright "Coders Guru"
#property link      "http://www.forex-tsd.com"
#define MAGICMA  20060306



//---- Trades limits
extern double    TakeProfit=180;
extern double    TrailingStop=30;
extern double    StopLoss=70;
extern bool      UseStopLoss = false;

extern double    HedgingTakeProfit=20;
extern double    HedgingStopLoss=10;
extern bool      UseHedging = true;
extern bool      ContinuesHedging = true;


//---- EMAs paris
extern int ShortEma = 10;
```

```
extern int LongEma = 80;

//---- Crossing options
extern bool ImmediateTrade = true; //Open trades immediately or wait for cross.
extern bool CounterTrend = true; //Use the originally CounterTrend crossing method or not

//---- Money Management
extern double Lots = 1;
extern bool UseMoneyManagement = true; //Use Money Management or not
extern bool AccountIsMicro = false; //Use Micro-Account or not
extern int Risk = 10; //10%

//---- Time Management
extern bool    UseHourTrade = false;
extern int     FromHourTrade = 8;
extern int     ToHourTrade = 18;

extern bool Show_Settings = true;
extern bool  Summarized = false;
extern double slippage = 3;


//+------------------------------------------------------------------+
//| expert initialization function                                   |
//+------------------------------------------------------------------+
int init()
  {
//----
   if(Show_Settings && Summarized == false) Print_Details();
   else if(Show_Settings && Summarized) Print_Details_Summarized();
   else Comment("");
//----

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

bool isNewSumbol(string current_symbol)
  {
   //loop through all the opened order and compare the symbols
   int total  = OrdersTotal();
   for(int cnt = 0 ; cnt < total ; cnt++)
   {
      OrderSelect(cnt, SELECT_BY_POS, MODE_TRADES);
      string selected_symbol = OrderSymbol();
      if (current_symbol == selected_symbol)
      return (False);
    }
    return (True);
}

int Crossed()
 {
   double EmaLongPrevious = iMA(NULL,0,LongEma,0,MODE_EMA, PRICE_CLOSE, 1);
   double EmaLongCurrent = iMA(NULL,0,LongEma,0,MODE_EMA, PRICE_CLOSE, 0);
   double EmaShortPrevious = iMA(NULL,0,ShortEma,0,MODE_EMA, PRICE_CLOSE, 1);
   double EmaShortCurrent = iMA(NULL,0,ShortEma,0,MODE_EMA, PRICE_CLOSE, 0);

   if(ImmediateTrade)
   {
      if (EmaShortCurrent<EmaLongCurrent)return (1); //down trend
      if (EmaShortCurrent>EmaLongCurrent)return (2); //up trend
   }
```

```
   if (EmaShortPrevious>EmaLongPrevious && EmaShortCurrent<EmaLongCurrent ) return (1); //down
   if (EmaShortPrevious<EmaLongPrevious && EmaShortCurrent>EmaLongCurrent ) return (2); //up tr

   return (0); //elsewhere
 }


//--- Bassed on Alex idea! More ideas are coming
double LotSize()
{
     double lotMM = MathCeil(AccountFreeMargin() *  Risk / 1000) / 100;

     if(AccountIsMicro==false) //normal account
     {
        if (lotMM < 0.1) lotMM = Lots;
        if ((lotMM > 0.5) && (lotMM < 1)) lotMM=0.5; //Thanks cucurucu
        if (lotMM > 1.0) lotMM = MathCeil(lotMM);
        if  (lotMM > 100) lotMM = 100;
     }
     else //micro account
     {
        if (lotMM < 0.01) lotMM = Lots;
        if (lotMM > 1.0) lotMM = MathCeil(lotMM);
        if  (lotMM > 100) lotMM = 100;
     }

     return (lotMM);
}

string BoolToStr ( bool value)
{
   if(value) return ("True");
   else return ("False");
}
void Print_Details()
{
   string sComment = "";
   string sp = "----------------------------------------\n";
   string NL = "\n";

   sComment = sp;
   sComment = sComment + "TakeProfit=" + DoubleToStr(TakeProfit,0) + " | ";
   sComment = sComment + "TrailingStop=" + DoubleToStr(TrailingStop,0) + " | ";
   sComment = sComment + "StopLoss=" + DoubleToStr(StopLoss,0) + " | ";
   sComment = sComment + "UseStopLoss=" + BoolToStr(UseStopLoss) + NL;
   sComment = sComment + sp;
   sComment = sComment + "ImmediateTrade=" + BoolToStr(ImmediateTrade) + " | ";
   sComment = sComment + "CounterTrend=" + BoolToStr(CounterTrend) + " | " ;
   if(UseHourTrade)
   {
   sComment = sComment + "UseHourTrade=" + BoolToStr(UseHourTrade) + " | ";
   sComment = sComment + "FromHourTrade=" + DoubleToStr(FromHourTrade,0) + " | ";
   sComment = sComment + "ToHourTrade=" + DoubleToStr(ToHourTrade,0) + NL;
   }
   else
   {
   sComment = sComment + "UseHourTrade=" + BoolToStr(UseHourTrade) + NL;
   }


   sComment = sComment + sp;
   sComment = sComment + "Lots=" + DoubleToStr(Lots,0) + " | ";
   sComment = sComment + "UseMoneyManagement=" + BoolToStr(UseMoneyManagement) + " | ";
   sComment = sComment + "AccountIsMicro=" + BoolToStr(AccountIsMicro) + " | ";
   sComment = sComment + "Risk=" + DoubleToStr(Risk,0) + "%" + NL;
   sComment = sComment + sp;

   Comment(sComment);
}

void Print_Details_Summarized()
{
```

```
   string sComment = "";
   string sp = "----------------------------------------\n";
   string NL = "\n";

   sComment = sp;
   sComment = sComment + "TF=" + DoubleToStr(TakeProfit,0) + " | ";
   sComment = sComment + "TS=" + DoubleToStr(TrailingStop,0) + " | ";
   sComment = sComment + "SL=" + DoubleToStr(StopLoss,0) + " | ";
   sComment = sComment + "USL=" + BoolToStr(UseStopLoss) + NL;
   sComment = sComment + sp;
   sComment = sComment + "IT=" + BoolToStr(ImmediateTrade) + " | ";
   sComment = sComment + "CT=" + BoolToStr(CounterTrend) + " | " ;
   if(UseHourTrade)
   {
   sComment = sComment + "UHT=" + BoolToStr(UseHourTrade) + " | ";
   sComment = sComment + "FHT=" + DoubleToStr(FromHourTrade,0) + " | ";
   sComment = sComment + "THT=" + DoubleToStr(ToHourTrade,0) + NL;
   }
   else
   {
   sComment = sComment + "UHT=" + BoolToStr(UseHourTrade) + NL;
   }


   sComment = sComment + sp;
   sComment = sComment + "L=" + DoubleToStr(Lots,0) + " | ";
   sComment = sComment + "MM=" + BoolToStr(UseMoneyManagement) + " | ";
   sComment = sComment + "AIM=" + BoolToStr(AccountIsMicro) + " | ";
   sComment = sComment + "R=" + DoubleToStr(Risk,0) + "%" + NL;
   sComment = sComment + sp;

   Comment(sComment);
}
//+------------------------------------------------------------------+
//| expert start function                                            |
//+------------------------------------------------------------------+
int start()
  {
//----

   if (UseHourTrade)
    {
      if (!(Hour()>=FromHourTrade && Hour()<=ToHourTrade))
      {
         Comment("Time for trade has not come yet!");
         return(0);
      }
   }

   int cnt, ticket, ticket2, total;

   string comment = "";
   if(CounterTrend==true) comment = "EMAC_Counter";
   if(CounterTrend==false) comment = "EMAC_Pro";
   if(ImmediateTrade==true) comment = comment + "_Immediate";
   if(ImmediateTrade==false) comment = comment + "Postponed";

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

   int isCrossed  = 0;
   isCrossed = Crossed ();

   if(CounterTrend==false)
```

```
{
   if(isCrossed==1) isCrossed=2;
   if(isCrossed==2) isCrossed=1;
}

if(UseMoneyManagement==true) Lots = LotSize(); //Adjust the lot size

total  = OrdersTotal();

if(total < 1 || isNewSumbol(Symbol()))
  {
    if(isCrossed == 1)
      {

         if(UseStopLoss)
            ticket=OrderSend(Symbol(),OP_BUY,Lots,Ask,slippage,Ask-StopLoss*Point,Ask+TakePr
         else
            ticket=OrderSend(Symbol(),OP_BUY,Lots,Ask,slippage,0,Ask+TakeProfit*Point,commen

         if(ticket>0)
           {
            if(OrderSelect(ticket,SELECT_BY_TICKET,MODE_TRADES)) Print("BUY order opened : "
           }
         else Print("Error opening BUY order : ",GetLastError());

         if(UseHedging) Hedge(ticket,Lots);

         return(0);
      }
      if(isCrossed == 2)
      {
         if(UseStopLoss)
            ticket=OrderSend(Symbol(),OP_SELL,Lots,Bid,slippage,Bid+StopLoss*Point,Bid-TakeP
         else
            ticket=OrderSend(Symbol(),OP_SELL,Lots,Bid,slippage,0,Bid-TakeProfit*Point,comme

         if(ticket>0)
           {
            if(OrderSelect(ticket,SELECT_BY_TICKET,MODE_TRADES)) Print("SELL order opened :
           }
         else Print("Error opening SELL order : ",GetLastError());

         if(UseHedging) Hedge(ticket,Lots);

         return(0);
      }
      return(0);
  }



for(cnt=0;cnt<total;cnt++)
  {
   OrderSelect(cnt, SELECT_BY_POS, MODE_TRADES);

   if(OrderType()<=OP_SELL && OrderSymbol()==Symbol() && OrderMagicNumber() == MAGICMA)
     {
      if(OrderType()==OP_BUY)   // long position is opened
        {
         // check for trailing stop
         if(TrailingStop>0)
           {
            if(Bid-OrderOpenPrice()>Point*TrailingStop)
              {
               if(OrderStopLoss()<Bid-Point*TrailingStop)
                 {
                  OrderModify(OrderTicket(),OrderOpenPrice(),Bid-Point*TrailingStop,OrderTak
                  return(0);
                 }
              }
           }
```

```
            }
          else // go to short position
            {
             // 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,OrderTak
                      return(0);
                     }
                  }
               }
            }
         }
      }

   if(ContinuesHedging && OrdersCount(Symbol()) < 2)
     {
      //Print(OrdersCount(Symbol()));
      for(cnt=0;cnt<total;cnt++)
      {
          ticket = OrderSelect(cnt, SELECT_BY_POS, MODE_TRADES);
          if(OrderSymbol() == Symbol())
          Hedge(OrderTicket(),OrderLots());
      }
     }

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

int OrdersCount(string symbol)
{
   int total  = OrdersTotal();
   int count =0;
   for(int cnt = 0 ; cnt < total ; cnt++)
   {
      OrderSelect(cnt, SELECT_BY_POS, MODE_TRADES);
      if (symbol == OrderSymbol())
      count++;
    }
    return (count);
}

void Hedge (int order_ticket , double hlots)
{
   int ticket, order_type;
   double order_profit;
   string hcomment = "EMAC_" + Symbol() + "_Hedging";

   if(OrderSelect(order_ticket,SELECT_BY_TICKET,MODE_TRADES))
   {
      order_type = OrderType();
      order_profit = OrderProfit();

      if(order_type == OP_SELL && order_profit < 0 - slippage)
      {
            ticket=OrderSend(Symbol(),OP_BUY,hlots,Ask,slippage,Ask-HedgingStopLoss*Point,Ask+H

            if(ticket>0)
              {
               if(OrderSelect(ticket,SELECT_BY_TICKET,MODE_TRADES)) Print("Hedgin BUY order ope
              }
            else Print("Error opening Hedgin BUY order : ",GetLastError());
      }

      if(order_type == OP_BUY && order_profit < 0 - slippage)
      {
```

```
                ticket=OrderSend(Symbol(),OP_SELL,hlots,Bid,slippage,Bid+HedgingStopLoss*Point,Bid-

                if(ticket>0)
                  {
                   if(OrderSelect(ticket,SELECT_BY_TICKET,MODE_TRADES)) Print("Hedgin SELL order op
                  }
                else Print("Error opening Hedgin SELL order : ",GetLastError());
          }

      }
   }
```

Examples: ProfitProtector Expert Advisor!

Hi folks,

Today we are going to take an idea to create Expert Advisor (That the Expert Advisors are for).

The idea:

How to protect my profits of all opening orders (if there's any) by the mean I want to take them when they reach a specific amount and close all the trades when the drop down to a specific amount.

For example: There are 4 opened positions and they make profit. They've made 200 Pip right now. suddenly the profit went to 150 then 100 then I'm losing money.

Our code today will prevent this situation, let's see the idea programicaly.

The problem:

First we want a way to calculate the profit of all opened positions.

When the profit goes to specific amount we are going to close all the opened position. But, we want the program too to prevent the profit to drop again. So, we want to set another profit point that the program will prevent any drops below it.

For example: the program will take the profit when it reaches 200 Pips and prevent the profit to drop below to 100 Pips .

The programming problem now is when to start preventing the profit drop. How to tell the program that the profit went over 100 Pip and we don't want it drop back?

We need a third point when the profit reach it the program will start to monitor the profit drop.

The example right now is: the program will take the profit when it reaches 200 Pips and start to monitor the drop down of the profit when the profit reaches 150 Pips and prevent the profit to drop again to 100 Pips level.

Let's convert that to code!

The code:

Please download the code. I've added the profit protector code to a normal expert advisor of mine. We will not study the expert advisor strategy in buying/selling/closing/trailing. But we are going to study the code aimed to protect the profit.

These are the pieces of code concerning our idea:

```
.....//your normal code

extern double    ProfitToProtect = 200;
extern double    ProtectStarter = 150;
extern double    LossToProtect = 100;
extern bool      ProtectProfit= true;

bool      ProtectLoss= false;

.....//your normal code

int start()
{
...... //your normal code

if(ProtectProfit)
  ProfitProtect(ProfitToProtect);

  if(ProtectLoss)
  LossProtect(LossToProtect);
.......//your normal code
}

void ProfitProtect(double profit)
{
    int total  = OrdersTotal();
    double MyCurrentProfit=0;
    for (int cnt = 0 ; cnt < total ; cnt++)
    {
      OrderSelect(cnt,SELECT_BY_POS,MODE_TRADES);
      if (OrderMagicNumber() == MagicNumber)
        MyCurrentProfit += OrderProfit();
    }

    if(MyCurrentProfit>=ProtectStarter) //start protection at this level!
      ProtectLoss=true;
    if(MyCurrentProfit>=profit)
      CloseAll();
}

void LossProtect(double profit)
{
    int total  = OrdersTotal();
    double MyCurrentProfit=0;
    for (int cnt = 0 ; cnt < total ; cnt++)
    {
      OrderSelect(cnt,SELECT_BY_POS,MODE_TRADES);
```

```
      if (OrderMagicNumber() == MagicNumber)
        MyCurrentProfit += OrderProfit();
    }
    if(MyCurrentProfit<=profit)
      CloseAll();
}

void CloseAll()
{
    int total  = OrdersTotal();
    for (int cnt = 0 ; cnt < total ; cnt++)
    {
      OrderSelect(0,SELECT_BY_POS,MODE_TRADES);
      if (OrderMagicNumber() == MagicNumber)
        if(OrderType()==OP_BUY)
         OrderClose(OrderTicket(),OrderLots(),Bid,Slippage,  Violet);
        if(OrderType()==OP_SELL)
         OrderClose(OrderTicket(),OrderLots(),Ask,Slippage,  Violet);
    }
}
```

Code study:

We have defined our profit point as external variables at the top of the program:

```
extern double    ProfitToProtect = 200;
extern double    ProtectStarter = 150;
extern double    LossToProtect = 100;
extern bool      ProtectProfit= true;
```

Where the the ProfitToProtect is the amount we want the program to take the profit when it is reached. And the ProtectStarter is the amount the program start at monitoring the profit to prevent drop down again. And finally the LossToProtect is the amount of the profit the program prevent the profit to drop below it.

We have added the option to the user to use our profit protector or not with the variable ProtectProfit;

```
bool      ProtectLoss= false;
```

We have declared this variable outside the start() function to make it in a global scope and we will use this variable to start the profit monitoring.

```
if(ProtectProfit)
  ProfitProtect(ProfitToProtect);
```

If the user has chosen to use our protect (ProtectProfit=True) then the program will call the function ProfitProtect(). We are going to study this function soon.

```
if(ProtectLoss)
  LossProtect(LossToProtect);
```

If the ProtectLoss variable is true (It will be true when the profit reaches the ProtectStarter level) then the program will call the function LossProtect() which we are going to study it soon.

```
void ProfitProtect(double profit)
```

```
{
    int total  = OrdersTotal();
    double MyCurrentProfit=0;
    for (int cnt = 0 ; cnt < total ; cnt++)
    {
      OrderSelect(cnt,SELECT_BY_POS,MODE_TRADES);
      if (OrderMagicNumber() == MagicNumber)
        MyCurrentProfit += OrderProfit();
    }

    if(MyCurrentProfit>=ProtectStarter) //start protection at this level!
      ProtectLoss=true;
    if(MyCurrentProfit>=profit)
      CloseAll();
}
```

**The ProfitProtect() function takes a double parameter (profit you want to protect).
It starts by getting the number of opened orders by calling the function OrdersTotal()
then it declares a double variable MyCurrentProfit and initialize it to 0. We will use this variable to calculate the profit of the opened positions.**

Now we enter in loop from 0 to the count of the opened positions. In every loop we select the order using OrderSelect() function and examine is the MagicNumber of the order is the same as our MagicNumber variable to be sure that we are working only with the positions have been opened with our Expert Advisor. Then we get the profit of the order using OrderProfit() and add it to the MyCurrentProfit variable.

Now we have the total profit stored in the MyCurrentProfit variable, We can check it now to find did it reach the ProtectStarter value; if true we will enable ProtectLoss. And we have to check the MyCurrentProfit variable against ProfitToProtect variable to find did it reach it or not; if true we will call the CloseAll() function which will close all the opened positions.

```
void LossProtect(double profit)
{
    int total  = OrdersTotal();
    double MyCurrentProfit=0;
    for (int cnt = 0 ; cnt < total ; cnt++)
    {
      OrderSelect(cnt,SELECT_BY_POS,MODE_TRADES);
      if (OrderMagicNumber() == MagicNumber)
        MyCurrentProfit += OrderProfit();
    }
    if(MyCurrentProfit<=profit)
      CloseAll();
}
```

The LossProtect() function is very like the ProfitProtect() function, the only difference is it will monitor the MyCurrentProfit value and close all the opened position if it goes below the LossToProtect vlaue.

```
void CloseAll()
{
    int total  = OrdersTotal();
    for (int cnt = 0 ; cnt < total ; cnt++)
    {
      OrderSelect(0,SELECT_BY_POS,MODE_TRADES);
      if (OrderMagicNumber() == MagicNumber)
        if(OrderType()==OP_BUY)
         OrderClose(OrderTicket(),OrderLots(),Bid,Slippage,  Violet);
```

```
        if(OrderType()==OP_SELL)
         OrderClose(OrderTicket(),OrderLots(),Ask,Slippage,  Violet);
     }
}
```

The CloseAll() function goes through all the opened position and check them against our expert MagicNubmer then close every buy or sell position using OrderClose() function.

I hope everything is clear, And hope you find it a useful idea!

Coders Guru

Global variables

Hi folks,

Today we are going to study a set of MQL4 functions which needs to be clearer for the new comers to the language. We are going to study the Global variables and the MQL4 functions that handle them.

What are the global variables?

The global variables are places to store data between the instances of MQL4 programs and MetaTrader launches too.

In MQL4 you use the normal variables to store the data temporary and it exists only at the time that program is running and vanishes with the un-initialization of the program. But the global variables exist when you un-initialize the program and even if you shut down the terminal itself. The terminal can store the global variables for four weeks from the last store.

How useful the global variables?

The main job of the global variables is enabling the inner communication between the experts advisor.

Assume that you have created two expert advisors:

The first expert advisor works on daily time frame and buys the EURUSD when the Moving Averages of the price of the last 10 days crosses upward the Moving Averages of the price of the last 80 days.

And it sells the EURUSD when the Moving Averages of the price of the last 10 days crosses downward the Moving Averages of the price of the last 80 days.

And the second expert advisor works on 1 hour time frame and buys the EURUSD when the Moving Averages of the price of the last 10 hours crosses upward the Moving Averages of the price of the last 80 hours.

And it sells the EURUSD when the Moving Averages of the price of the last 10 hours crosses downward the Moving Averages of the price of the last 80 hours.

You don't want the two expert advisors to take conflicted decisions. You don't want the first expert advisor to sell the EURUSD while the second expert advisor is buying the EURUSD.

Any of the two expert advisors doesn't know anything about the other. How can they communicate?

They can communicate with the aid of writing/reading global variables.

For instance the first expert advisor can write a global variable when it open sell/buy order while the second expert advisor can read this variable to be sure that it will not open a conflicting orders.

Of course the possibilities of using global variables are not limited. Our example today will use the global variables in a very useful way.

Our example:

When you attach an expert advisor to more than one chart/pair and want to change one of the expert advisor inputs values for all the charts you have to change this input for every chart.

If your expert advisor attached to ten pairs it will be a real problem to make the change 10 times.

Our program today will use the global variables to make the changes available to all the charts hold our expert advisor.

Global variables functions:

There are 6 functions in MQL4 responsible of handling the global variables. These are the global variables functions:

GlobalVariableCheck:

Syntax:

bool **GlobalVariableCheck** (string **name**)

Description:

The *GlobalVariableCheck* function checks if there's a global variable with the name passed to, it returns true if it exists and false if not exists.

You can use *GetLastError* to get the error information if any error has been occurred will you calling this function.

Parameters:

This function takes only one parameter:

string name

The global variable name the function will check is it exists or not.

Example:

```
if(!GlobalVariableCheck("ge_TakeProfit "))

Alert("ge_TakeProfit global variable is not exist");
```

GlobalVariableDel:

Syntax:

```
bool GlobalVariableDel (string name)
```

Description:

The *GlobalVariableDel* function deletes the global variable you passed its name and return true in success and false if it failed to delete it.

You can use *GetLastError* to get the error information if any error has been occurred will you calling this function.

Note: If the global variable not found the returned value will be 0 while the error code will be 4057 which means global variables processing error.

Parameters:

This function takes only one parameter:

string name

The global variable name the function will delete.

Example:

```
Alert(GlobalVariableDel("ge_TakeProfit "));
```

GlobalVariableGet:

Syntax:

```
double GlobalVariableGet (string name)
```

Description:

The *GlobalVariableGet* function returns the value of the global variable name passed to it. This value must to be double data type.

You can use *GetLastError* to get the error information if any error has been occurred will you calling this function.

Note: You can store in the global variable and data type but the double data type. And if you tried to store any data type other than the double data type, MQL4 will try to convert it to double data type.

Note: If the global variable not found the returned value will be 0 while the error code will be 4058 which means global variable not found. So, you have to use GlobalVariableCheck function before using GlobalVariableGet if your variable maybe 0.

Parameters:

This function takes only one parameter:

string name

The global variable name the function will return its stored value.

Example:

```
if(GlobalVariableCheck("ge_TakeProfit"))

   TakeProfit = GlobalVariableGet("ge_TakeProfit");
```

GlobalVariableSet:

Syntax:

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

Description:

The *GlobalVariableSet* function sets the variable name passed to it with the value passed to it.

If the global variable name does not exist the function will create it and set it to the passed value. If it exists the function will change its old value to the new one.

The returned value is a datetime data type which is the time of the last access time of the global variable if the function successes and if it failed the return value will be 0.

You can use *GetLastError* to get the error information if any error has been occurred will you calling this function.

Parameters:

This function takes two parameters:

string name

The global variable name the function will set its value (or create it and sets its value).

double value

The value of the global variable you want to set (or create). This value must be double (numeric) data type.

Example:

```
Alert(GlobalVariableDel("ge_TakeProfit "));
```

GlobalVariableDel:

Syntax:

```
bool GlobalVariableDel (string name)
```

Description:

The *GlobalVariableDel* function deletes the global variable you passed its name and return true in success

and false if it failed to delete it.

You can use *GetLastError* to get the error information if any error has been occurred will you calling this function.

Note: If the global variable not found the returned value will be 0 while the error code will be 4057 which means global variables processing error.

Parameters:

This function takes only one parameter:

string name

The global variable name the function will delete.

Example:

```
Alert(GlobalVariableDel("ge_TakeProfit "));
```

iMaOnArray (Moving average of indicator)

Hi folks,

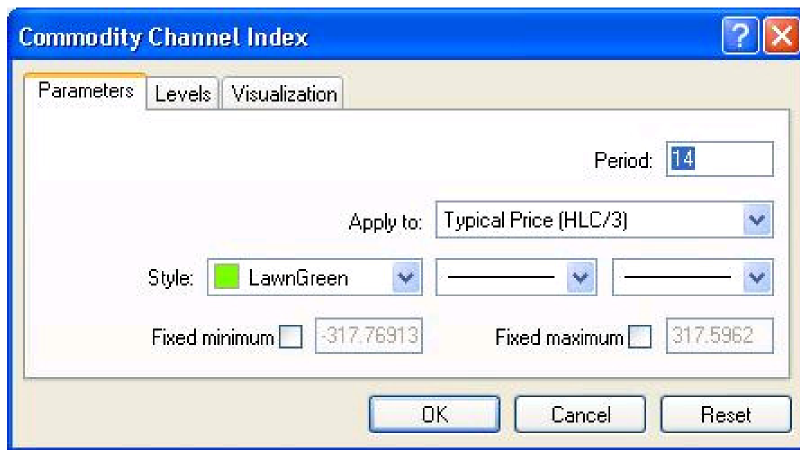
I've got a lot messages in the forex-tsd forum asking me how to use iMaOnArray function to get the Moving Average of a specific indicator. Today I'm going to answer all of you in this article.

What's the moving average of indicator?

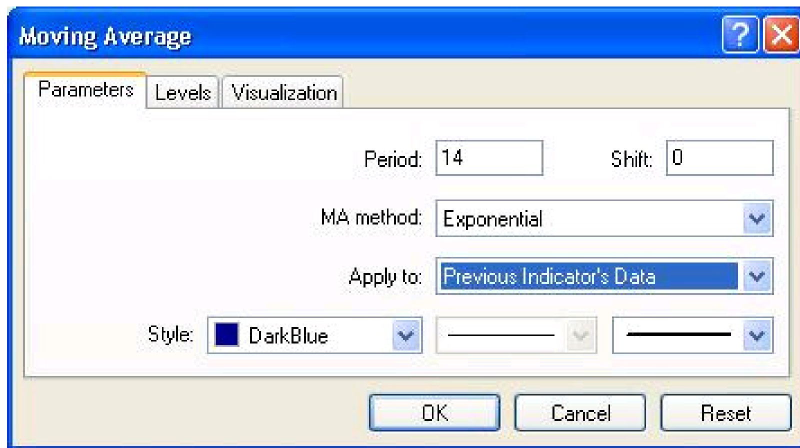
When you attach an indicator to the chart it uses the price to for it calculation and save (draw) them . You can calculate the moving average of this array.

I'll give you a manual example before going to create our mql4 version.

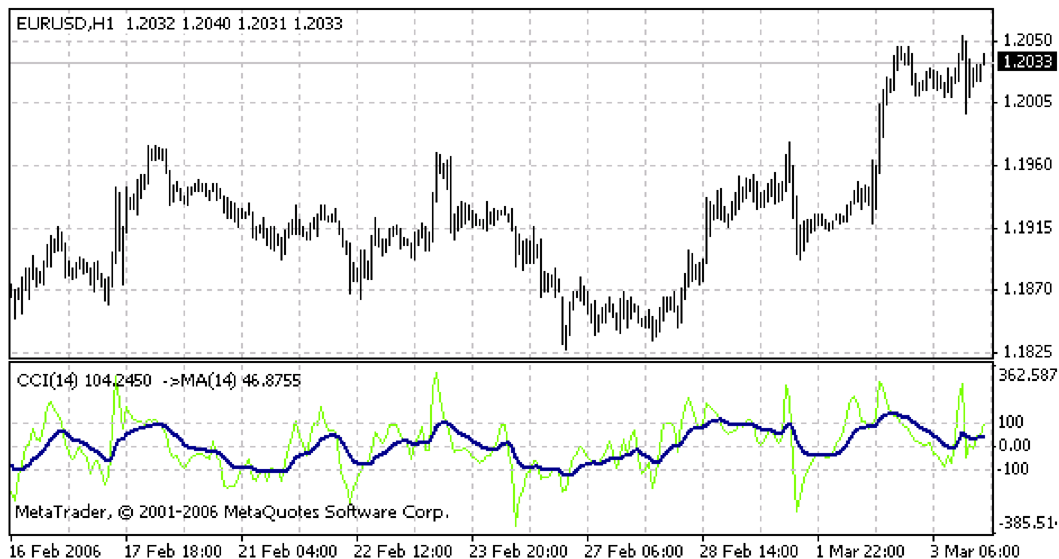
Let's say that we will attach the CCI (Commodity Channel Index) indicator to our chart (Figure 1).



Now we want to get the moving average of the CCI. We can do that by dragging the Moving Average indicator to the window holds the CCI indicator and choose the Apply to: Previous Indicator's Data (Figure 2).



And that's what you get (Figure 3)



How to do that programicly?

If you are interested in getting the moving average of any indicator like the above manual setup, you can 'trigger' your MetaEditor and write this code:

```
//+------------------------------------------------------------------+
//|                                                  iMAOnArray.mq4 |
//|                                                     Coders Guru |
//|                                         http://www.metatrader.info |
//+------------------------------------------------------------------+

#property copyright "Coders Guru"
#property link      "http://www.metatrader.info"


#property indicator_separate_window
#property indicator_buffers 2
#property indicator_color1 LawnGreen
#property indicator_color2 DarkBlue

double ExtMapBuffer1[];
double ExtMapBuffer2[];

int init()
  {
   IndicatorDigits(MarketInfo(Symbol(),MODE_DIGITS));
   SetIndexStyle(0,DRAW_LINE);
   SetIndexBuffer(0,ExtMapBuffer1);
   SetIndexStyle(1,DRAW_LINE,STYLE_SOLID,2);
   SetIndexBuffer(1,ExtMapBuffer2);

   return(0);
  }

int deinit()
  {
   return(0);
  }

int start()
  {
   int bar, limit;

   int counted_bars=IndicatorCounted();
   if(counted_bars<0) return(-1);
   if(counted_bars>0) counted_bars--;
   limit=Bars-IndicatorCounted();


   for(bar=0; bar<limit; bar++)
      ExtMapBuffer1[bar] = iCCI(NULL,0,14,PRICE_TYPICAL,bar);

   for(bar=0; bar<limit; bar++)
         ExtMapBuffer2[bar]=iMAOnArray(ExtMapBuffer1,Bars,14,0,MODE_EMA,bar);

   return(0);
  }
```

As you can notice in the above code that we use the function iCCI and iMAOnArray to calculate the Moving average of the CCI indicator.

The iCCI function simply calculate the CCI of a giving bar and return its value. We store those values returned by iCCI function in our buffer ExtMapBuffer1.

Now we have a full of data buffer (ExtMapBuffer1), how to get the Moving Average of this buffer?

iMAOnArray:

With the aid of the iMAOnArray function we can calculate the moving average of the values stored in an array(buffer).

This is the syntax of the iMAOnArray function:

double iMAOnArray(double array [],int total,int period,int ma_shift,int ma_method,int shift)

The iMAOnArray - as you see in its syntax - returns double value; this is the value of the moving average - counted on the array - of the given bar. Don't worry you will understand well when you know the parameters of the iMAOnArray function.

Parameters:

double array []:

This is the array of the values you want to calculate the moving average of it.
You have to fill this array with double data type items.
In our example we used the bufferex1 as the array[] parameter passed to iMAOnArray after filling that array with the values of the CCIs of the bars in our chart.

int total:

You use the total parameter to indicate the count of items of the data from the array you want to use to calculate the moving average.
If you want to use all the items in the array in your moving average calculation pass 0 in the total parameter.

int period:

The moving average period of the moving average you want to use.
If you are familiar to moving average you can skip this section.
when use the moving average indicator with the hourly chart and use 12 as the period of moving average calculation ; it means that you want to know the average of the price of the previous 12 hours.
when use the moving average indicator with the daily chart and use 30 as the period of moving average calculation ; it means that you want to know the average of the price of the previous 30 days.

int ma_method:

The moving average method you want to use in your calculation.
These are the moving average methods available in MetaTrader

Lesson 10 - Your First Indicator (Part1)

Welcome to the practical world of MQL4 courses; welcome to your first indicator in MQL4.

I recommend you to read the previous nine lessons very carefully, before continuing with these series of courses, that’s because we will use them so much in our explanations and studies of the Expert Advisors and Custom Indicators which we will create in this series of lessons.

Today we are going to create a simple indictor which will not mean too much for our trade world but it means too much to our MQL4 programming understanding.

It simply will collect the subtraction of High [] of the price – Low [] of the price; don't be in a hurry, you will know everything very soon.

Let's swim!

MetaEditor:

This is the program which has been shipped with MT4 (MetaTrader 4) enables you to write your programs, read MQL4 help, compile your program and More.

I've made a shortcut for MetaEditor on my desktop for easily access to the program.
If you want to run MetaEditor you have three choices.

1- Run MT4, then click F4, choose MetaEditor from Tools menu or click its icon on the Standard toolbar (Figure 1).

2- From Start menuà Programs, find MetaTrader 4 group then click MetaEditor.

3- Find the MT4 installation path (usually C:\Program Files\MetaTrader 4), find the MetaEditor.exe and click it (I recommend to make a shortcut on you desktop).

Figure 1 – MetaTrader Standard Toolbar

Any method you have chosen leads you to MetaEditor as you can see in figure 2.

As you can see in figure 2, there are three windows in MetaEditor:

1- The Editor window which you can write your program in.

2- The Toolbox window which contains three tabs:

- **a. Errors tab, you see here the errors (if there any) in your code.**
- **b. Find in files tab, you see here the files which contain the keyword you are searching for using the toolbar command "Find in files" or by clicking CTRL +SHIFT+ F hotkeys.**
- **c. Help tab, you can highlight the keyword you want to know more about it and click F1, and you will see the help topics in this tab.**

3- The Navigator window which contains three tabs:

- **a. Files tab, for easy access to the files saved in the MT4 folder.**
- **b. Dictionary tab enables you to access the MQL4 help system.**

c. **Search tab enables you to search the MQL4 dictionary.**

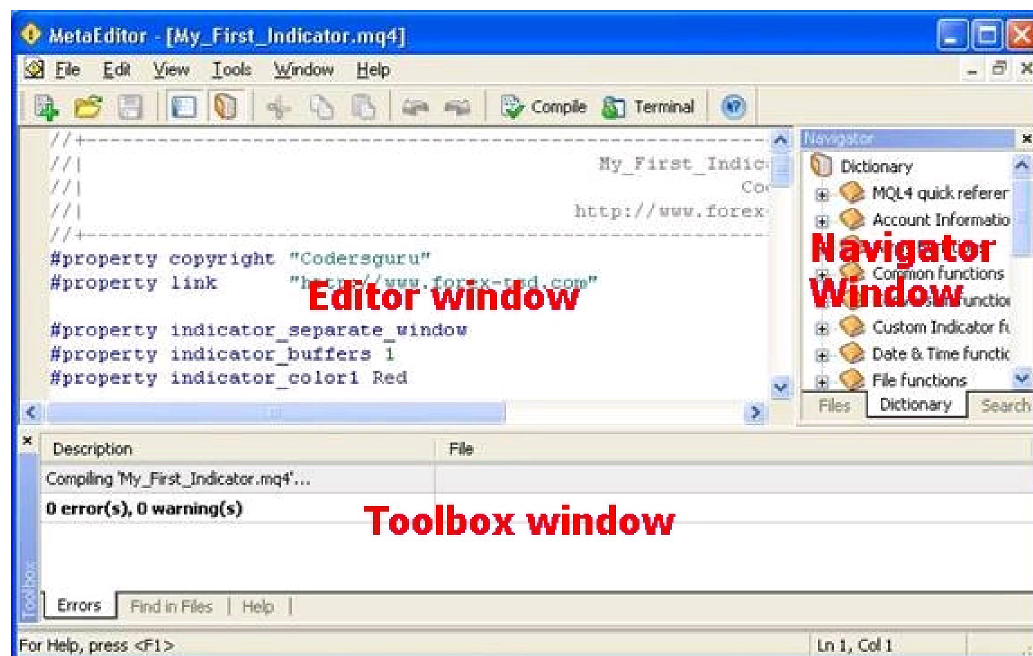


Figure 2 – MetaEditor Windows

I recommend you to navigate around the MetaEditor Menus, Toolbar and windows to be familiar with it.

Now let's enjoy creating our first custom indicator.

Note:Custom Indicator is a program which enables you to use the functions of the technical indicators and it cannot automate your deals.

First three steps:

Now you have run your MetaEditor and navigated around its Menus, Toolbar and windows, let's USE it.

To create a custom indicator you have to start with three steps (you will learn later how to skip these boring steps (my personal opinion).

Step 1: Click File menu à New (you use CTRL+N hotkey or click the New Icon in the Standard toolbar).

You will get a wizard (Figure 3) guiding you to the next step.

Choose Custom Indicator Program option, and then click next.

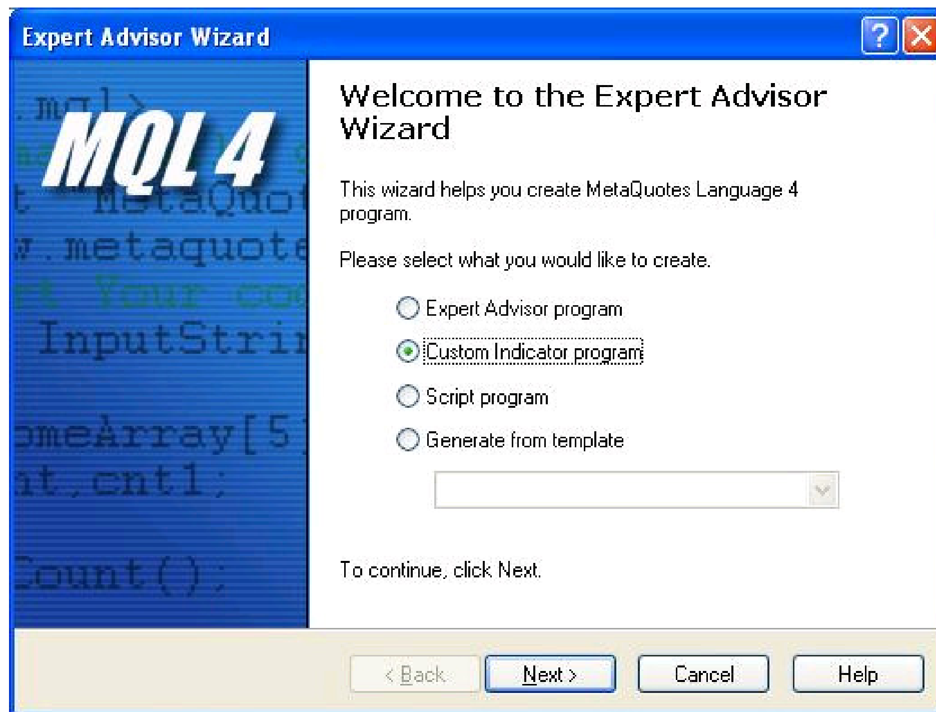


Figure 3 - New project wizard

Step 2: When you clicked Next, you will get the second step wizard (Figure 4) which will enable you to edit the properties of your program. In this step you can enter these properties:

1- Name of your program, this is the name which the world will call you program with and it will be saved as the_name_you_have_chosen.mq4

2- Author name, the creator of the program name.

3- Link to your web site.

4- External variables list: I want to pause here to remember you about external variable.

External variables are the variables which will be available to the user of you indicator to set from the properties tab of your indicator in MetaTrader. For example: MA_Period in the very popular EMA indicator. And these variables will be declared with the "extern" keyword in your code (Please review Variables lesson).

So, this section of the wizard enables you to add these kinds of variables.

In our first indicator example we will not need any external variables just write the values you see in figure 4 and let's go to step 3 by clicking Next button.

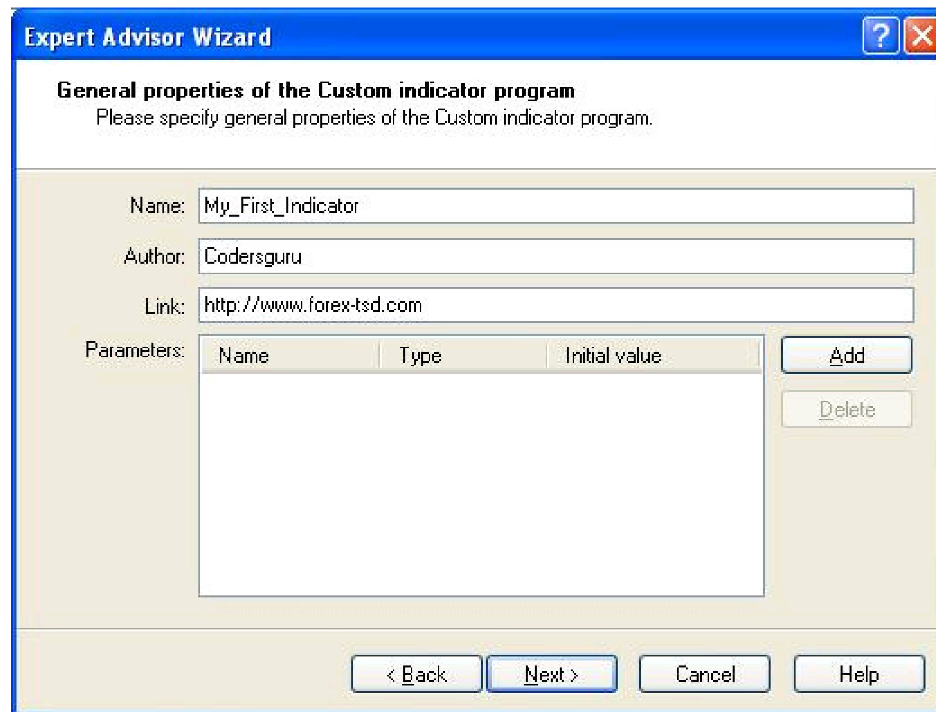


Figure 4 – Program properties wizard.

Step 3: The third wizard you will get when you clicked Next button is the Drawing properties wizard (Figure 5).

Its job is enabling you to set the dawning properties of the lines of your indicator, for example: how many lines, colors and where to draw your indicator (in the main chart or in separate windows).

This wizard contains the following options:

1- Indicator in separate window option: by clicking this option, your indicator will be drawn in separate windows and not on the main chart window. If you didn't check the option, your indicator will be drawn in the main chart window.

2- Minimum option: it will be available (enabled) only if you have checked the Indicator in separate window option, and its job is setting the bottom border for the chart.

3- Maximum option: it will be available (enabled) only if you have checked the Indicator in separate window option, and its job is setting the top border for the chart

4- Indexes List: here you add your indicator line and set its default colors.

I want you to wait to the next lesson(s) to know more about these options and don't be in a hurry.

For our first indicator example, choose Indicator in separate window option and click Add button, when you click add button the wizard will add a line to the indexes list like you see in figure 5.

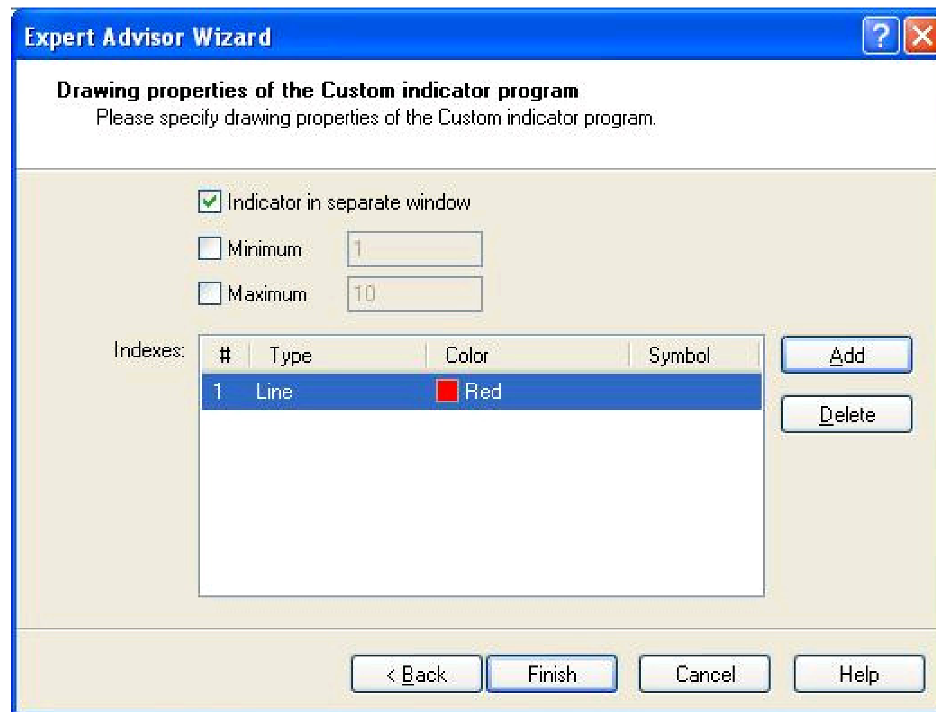


Figure 5 - Drawing properties wizard.

When you click Finish button the Magic will start. You will see the wizard disappeared and the focus returned to the MetaEditor environment and… guess what?

You have ready to use first indicator draft code.

This is the code you will get:

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

```
//+------------------------------------------------------------------+
//| Custor indicator deinitialization function |
//+------------------------------------------------------------------+
int deinit()
{
//----

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

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

As you see in the above code, the wizard has written a lot of code for you, now I have to thank the wizard and to thank you too.

In the next lesson we will discover every line of code you have seen above and add our code to make our first indicator. To this lesson I hope you be ready!

Please don't forget to download the source code of the first indicator and warm yourself for the next lesson.

I welcome very much the questions and the suggestions.

See you

Coders' Guru

Lesson 11 - Your First Indicator (Part2)

Welcome to the second part of "Your First Indicator" lesson.

In the previous lesson we didn't write any line of code, that's because the New Project

Wizard wrote all the code for us. Thanks!

Today we are going to add few lines to the code the wizard had generated to make our

program more useful.

Afterwards, we are going to explain the whole of the code line by line.

Let's coding

Code we have added:

We have added the code which in a bold dark blue to our previous 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);
}
//+------------------------------------------------------------------+
```

How will we work?

We will write the line(s) of the code we are going to explain then we will explain them

afterwards, if there are no topics, we will explain the line(s) of code directly. But at the most of the time we will pause to discuss some general topics.

I want to here your suggestion about this method please!

Now let's crack this code line by line.

```
//+------------------------------------------------------------------+
//| My_First_Indicator.mq4 |
//| Codersguru |
//| http://www.forex-tsd.com |
//+------------------------------------------------------------------+
```

Comments:

The first five lines of code (which are in gray color) are comments.

You use Comments to write lines in your code which the compiler will ignore them.

You are commenting your code for a lot of reasons:

_ To make it clearer

_ To document some parts like the copyright and creation date etc.

_ To make it understandable.

_ To tell us how the code you have written is work.

_ …

You can write comments in two ways:

Single line comments: The Single line comment starts with "//" and ends with the new line.

Multi-line comments: The multi-line comment start with "/*" and ends with "*/" and you can comment more than one line.

In our program the MQL4 wizard gathered from the data we entered the name of the program, author and the link and wrote them as comments at the top of our program.

```
#property copyright "Codersguru"
#property link "http://www.forex-tsd.com"
#property indicator_separate_window
#property indicator_buffers 1
#property indicator_color1 Red
```

Property directive:

As you notice all of these lines start with the word (#property). That's because they are kind of the Preprocessors directives called property directives.

The Preprocessors are the instructions you give to the compiler to carry them out before starting (processing) your code.

The property directives are predefined constants called "Controlling Compilation" built in the MQL4 language; their job is setting the properties of your program.

For example: is your Indicator will appear in the main chart window or in a separate window? Who is the writer of the program?

Note: The preprocessors lines end with a carriage-return character (new line) not a semi-colon symbol.

We will try to discuss here the property directives available in MQL4.

link:

This property setting the web link to your web site which you asked to enter it in step 2 in the Expert Advisor Wizard (review the previous lesson).

The data type of this property is string.

copyright:

It's the name of the author of the program, same as the link property you asked to enter it in step 2 in the Expert Advisor Wizard.

The data type of this property is string.

stacksize:

It's an integer value sets the memory size for every thread, the default value is 16384.

The data type of this property is integer.

indicator_chart_window:

When you set this property, your indicator will be drawn in the main chart window (Figure 1). You have to choose one of two options for your Indicators, drawing them in the main chart windows by using this property, or drawing them in separate windows by choosing the indicator_separate_window. You can't use the both of them at the same time.

The data type of this property is void, which means it takes no value.

indicator_separate_window:

When you set this property, your indicator will be drawn in a separate window (Figure

1). You can set the scale of the separate indicator window using two properties

indicator_minimum for the minimum value and indicator_maximum for the

maximum value of the scale.

And you can set the level of your indicators on these scales using the property

indicator_levelN where's the N is the indicator number.

Both of the properties indicator_chart_window and indicator_separate_window are

void data type, which mean they don't take value and you just write them.

In our program we will draw our indicator in a separate window:

#property indicator_separate_window

Figure 1

indicator_minimum:

With the aid of this property we are setting the minimum value of the separate windows

scale, which is the bottom border of the windows. For example:

#propery indicator_minimum 0

#propery indicator_ maximum 100

Here we have set the bottom border of the window to 0 and the top border to 100 (see

indicator_ maximum), hence we have a scale ranged from 0 to 100 in our separate

window which we are drawing our indicator.

The data type of this property is integer.

indicator_maximum:

With the aid of this property we are setting the maximum value of the separate windows

scale, which is the top border of the windows.

This value must be greater than the indicator_minimum value.

The data type of this property is integer.

Main chart window

Separate window

indicator_levelN:

With the aid of this property we are setting the level of the indicator in the scale we have created with the properties indicator_minimum and indicator_maximum, this value

must be greater than the indicator_minimum value and smaller than the

indicator_maximum value.

N is the indicator number which we are setting its level, it must range from 1 to 8 (because we are allowed only to use up to 8 indicator buffers in our program, so we can set the indicator_level for each of them using its number). For example:

#property indicator_minimum 0

#property indicator_minimum 100

#property indicator_level1 10 //set the first indicator level

#property indicator_level2 65.5 //set the second indicator level

The data type of this property is double.

indicator_buffers:

With the aid of this property we are setting the number of memories spaces (Arrays) allocated to draw our line(s). When we set the number (ranged from 1 up to 8) we are telling MQL4: "Please allocate a memory space for me to draw my indicator line".

In our program we used only one buffer.

#property indicator_buffers 1

That's because we will draw only one line.

indicator_colorN:

We can use up to 8 lines in our indicator, you can set the color of each of them using this property indicator_colorN , where the N is the line number which defined by indicator_buffers.

The user of your Indicator can change this color from the properties dialog of your Indicator (Figure 2).

In our program the indicator line color will be red.

#property indicator_color1 Red

The data type of this property is color.

Figure 2

double ExtMapBuffer1[];

Arrays:

In our life we usually group similar objects into units, in the programming we also need to group together the data items of the same type. We use Arrays to do this task.

Arrays are very like the list tables, you group the items in the table and access them the number of the row. Rows in the Arrays called Indexes.

To declare an array you use a code like that:

int my_array[50];

Here, you have declared an array of integer type, which can hold up to 50 items.

You can access each item in the array using the index of the item, like that:

My_array[10] = 500;

Here, you have set the item number 10 in the array to 500.

You can initialize the array at the same line of the declaration like that:

int my_array[5] = {1,24,15,66,500};

In our program we used this line of code:

double ExtMapBuffer1[];

Here we have declared and array of double type. We will use array to calculate our values which we will draw them on the chart.

```
int init()
{
}
```

Special functions:

Functions are blocks of code which like a machine takes inputs and returns outputs (Please review lesson 7 – Functions).

In MQL4 there are three special functions

init():

Every program will run this function before any of the other functions, you have to put here you initialization values of you variables.

start():

Here's the most of the work, every time a new quotation have received your program will call this function.

deinit():

This is the last function the program will call before it shutdown, you can put here any removals you want.

```
SetIndexStyle(0,DRAW_LINE);
SetIndexBuffer(0,ExtMapBuffer1);
string short_name = "Your first indicator is running!";
```

IndicatorShortName(short_name);

Custom indicator functions:

I can't give you a description for all of the indicators functions in this lesson. But we will use them all in our next lessons with more details. So, we will study here the functions used in our program.

SetIndexStyle:

void SetIndexStyle(int index, int type, int style=EMPTY, int width=EMPTY, color clr=CLR_NONE)

This function will set the style of the drawn line.

The index parameter of this function ranges from 1 to 7 (that's because the array indexing start with 0 and we have limited 8 line). And it indicte which line we want to set its style.

The type parameter is the shape type of the line and can be one of the following shape type's constants:

DRAW_LINE (draw a line)

DRAW_SECTION (draw section)

DRAW_HISTOGRAM (draw histogram)

DRAW_ARROW (draw arrow)

DRAW_NONE (no draw)

The style parameter is the pen style of drawing the line and can be one of the following styles' constants:

STYLE_SOLID (use solid pen)

STYLE_DASH (use dash pen)

STYLE_DOT (use dot pen)

STYLE_DASHDOT (use dash and dot pen)

STYLE_DASHDOTDOT (use dash and double dots)

Or it can be EMPTY (default) which means it will be no changes in the line style.

The width parameter is the width of line and ranges from 1 to 5. Or it can be EMPTY (default) which means the width will not change.

The clr parameter is the color of the line. It can be any valid color type variable. The default value is CLR_NONE which means empty state of colors.

In our line of code:

SetIndexStyle(0,DRAW_LINE);

We have set the index to 0 which means we will work with the first (and the only) line.

And we have set the shape type of our line to DRAW_LINE because we want to draw a line in the chart.

And we have left the other parameters to their default values.

SetIndexBuffer:

bool SetIndexBuffer(int index, double array[])

This function will set the array which we will assign to it our indicator value to the indicator buffer which will be drawn.

The function takes the index of the buffer where's 0 is the first buffer and 1 is the second, etc. Then it takes the name of the array.

It returns true if the function succeeds and false otherwise.

In our program the array which will hold our calculated values is ExtMapBuffer1.

And we have only one indicator buffer (#property indicator_buffers 1). So it will be the buffer assigned.

IndicatorShortName:

void IndicatorShortName(string name)

This function will set the text which will be showed on the upper left corner of the chart window (Figure 3) to the text we have inputted.

In our program we declared a string variable and assigned the value "You first indicator is running" to it, then we passed it to the IndicatorShortName function.

string short_name = "Your first indicator is running!";

IndicatorShortName(short_name);

Figure 3

return(0);

This is the return value of the init() function which terminate the function and pass the program to the execution of the next function start().

int deinit()

{

//----

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

Nothing new to say about deinit() function.

We will continue with remaining of the code in the next lesson.

I hope you enjoyed the lesson and I welcome your questions.

See you

Coders' Guru

Lesson 16 - Your First Expert Advisor (Part 4)

we have reached the edge of the moon in our way to the truth, I'm not under the impact of alcoholic poisoning (I don't drink at all), but I'm so happy to reach the last part of explaining or first expert advisor. Yes! This is the last part of the expert advisor lesson.

I hope you enjoyed the journey discovering how to write our simple yet important expert advisor.

Let's take the final step.

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);
}
//+------------------------------------------------------------------+
```

In the previous lesson, we checked the *OrdersTotal* is less than 1 in order to open a Buy or a Sell orders in the case that there were no already opened orders.

We have used this code:

```
if(total < 1)
{
if(isCrossed == 1)
{
.....
}
if(isCrossed == 2)
{
.....
}
return(0);
}
```

This was the *Open New Order routine*. Today we will study *Modify-Close Opened Orders routine*.

```
for(cnt=0;cnt<total;cnt++)
{
....
}
```

In the above block of code we used a *for* loop to go through all the already opened orders. We start the loop from the *cnt* = 0 and the end of the loop is the *total* number of already orders. Every loop cycle we increase the number of *cnt* by 1 (*cnt*++).So, *cnt* will hold in every cycle the poison of the order (0,1,2,3 etc) which we will use with *OrderSelect* function to select each order by its position.

Our today's mission is studying what's going inside the heart of the above loop.

```
OrderSelect(cnt, SELECT_BY_POS, MODE_TRADES);
if(OrderType()<=OP_SELL && OrderSymbol()==Symbol())
{
....
}
```

The *OrderSelect* function used to select an opened order or a pending order by the ticket number or by index.

We used the *OrderSelect* here before using the *OrderType* and *OrderSymbol* functions because if we didn't use *OrderSelect,* the *OrderType* and *OrderSymbol* functions will not work.

Note: You have to use *OrderSelect* function before the trading functions which takes no parameters:

OrderMagicNumber, OrderClosePrice, OrderCloseTime, OrderOpenPrice, OrderOpenTime, OrderComment, OrderCommission, OrderExpiration, OrderLots, OrderPrint, OrderProfit, OrderStopLoss, OrderSwap, OrderSymbol, OrderTakeProfit, OrderTicket and OrderType

We used *SELECT_BY_POS* selecting type which means we want to select the order by its index (position) not by its ticket number.

Note: The index of the first order is 0 and the index of the second one is 1 index etc.

And we used *MODE_TRADES* mode which means we will select from the currently trading orders (opened and pending orders) not from the history.

The *OrderType* function returns the type of selected order that will be one of:

***OP_BUY, OP_SELL, OP_BUYLIMIT, OP_BUYSTOP, OP_SELLLIMIT* or**

OP_SELLSTOP

We checked the type of the order to find is it *equal* or *lesser* than *OP_SELL*.

Which means it maybe one of two cases: *OP_SELL* or *OP_BUY* (because *OP_SELL*=1 and *OP_BUY* = 0). We did that because we will not work with pending orders.

We want too to work only with the order opened in the chart we loaded our expert advisor on, so we check the *OrderSymbol* of the order with the return value of *Symbol* function which returns the current chart symbol. If they are equal it means we are working with the currently loaded symbol.

So, all the coming code will work only if the *OrderType* is *OP_SELL* or *OP_BUY* and the *Symbol = OrderSymbol*.

```
if(OrderType()==OP_BUY) // long position is opened

{

....

}
```

We are working only with two types of orders, the first type is *OP_BUY*.

The code above means:

Is there a long (Buy) position opened? If yes! Execute this block of code….

Let's see what will do in the case of a long position has been opened

```
if(isCrossed == 2)

{

OrderClose(OrderTicket(),OrderLots(),Bid,3,Violet);

// close position

return(0); // exit

}
```

We have opened a Buy order when the *shortEma* crossed the *longEma* upward.

It's a logical to close this position when the *shortEma* and *longEma* crosses each others in reversal direction (downward).

So, we checked the *isCrossed* to find is it = 2 which means the reversal has been occurred and in this case we close the Buy order.

We used the *OrderClose* function to close the order. *OrderClose* function closes a specific opened order by its ticket. (Review appendix 2).

We've got the ticket number of the selected order using the *OrderTicket* function and passed it as the first parameter for *OrderClose* function.

The second parameter in the *OrderClose* is *Lots* (the number of lots); we used the *OrderLots* function to get the lots value of the selected order.

The third parameter in *OrderClose* is the preferred close price and we used the *Bid* function to get the bid price of the selected order.

The fourth parameter is the *slippage* value and we used 3.

The fifth parameter is the color of the closing arrow and we used *Violet* color.

We didn't forget to terminate the *start* function with *return(0)* statement.

```
// 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);
}
}
}
```

Note: We are still inside the block of: *if(OrderType()==OP_BUY)*.

We are going to apply our trailing stop technique for the opened Buy position in this block of code.

Firstly, we have checked the *TrailingStop* variable the user supplied to check was it a valid value or not (greater than 0).

Then we applied our trailing stop technique for the opened Buy orders which is:

We modify the *stoploos* of the order when the subtraction of the current bid price and

the opened price of order is greater than the *TrailingStop*

and

the current *stoploss* is lesser than the subtraction of the current bid price and the *TrailingStop*.

We used the *OrderModify* function to make the desired modification.

These are parameters we used with *OrderModify*:

ticket: We've got the current order ticket with *OrderTicket* function.

price: We've got the open price of the order with *OrderOpenPrice* function.

stoploss: Here's the real work! Because we are in a Buy position we set our new *stoploss* to the value of the subtraction of the current bid price and the *TrailingStop*.

That's our way trailing the *stoploss* point every time we make profits.

Note: Stop losses are always set BELOW the current bid price on a buy and ABOVE the current asking price on a sell.

takeprofit: No changes, we've got the current profit value of the order with *OrderTakeProfit* function.

expiration: We didn't set an *expiration* date to our order, so we used 0.

arrow_color: Still *Green* color.

Finally we terminate the start function.

```
else // go to short position
{
....
}
```

Note: *else* here belongs to the code:

```
if(OrderType()==OP_BUY) // long position is opened
{
....
}
```

We have studied the case of the type of the order is a Buy order.

So, we are working now a Sell order type.

Let's see what are we going to do in the case of a short (Sell) position has been already opened?

```
if(isCrossed == 1)
{
OrderClose(OrderTicket(),OrderLots(),Ask,3,Violet);
// close position
return(0); // exit
}
```

We've opened a Sell order when the *shortEma* crossed the *longEma* downward.

It's the time to close this position when the *shortEma* and *longEma* crosses each others in reversal direction (upward). Which happens in the case of *isCrossed* = 1.

We used the *OrderClose* function to close the order, we used the same parameters we use in the case of closing a Buy order except the third parameters the preferred close price, in this case is the Ask price.

Then we terminated the *start* function.

```
// 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);
}
}
}
```

We are going to apply our trailing stop technique for the opened Sell position in this block of code.

Firstly, we've checked the *TrailingStop* variable the user supplied to check was it a valid value or not (greater than 0).

Then we applied our trailing stop technique for the opened Sell orders which is:

We modify the *stoploss* of the order when the subtraction of the order's opened price

and the current ask price is greater than the *TrailingStop*

and

the current *stoploss* is greater than the addition of the current ask price and the

***TrailingStop*.**

We used the *OrderModify* function to make the desired modification. And used the same

parameters we use in the case of modifying already opened Buy order except the third

parameter which indicates our *stoploss* value:

We set our new *stoploss* to the value of the addition of the current ask price and the

***TrailingStop*.**

And the fifth parameter which indicates the color of the arrow in this case is *Red*.

Then we terminated the *start* function using *return(0)*;

return(0);

This line terminates the *start* function in all other case; there are no conditions to open

new positions and there are no needs to close or modify the already opened orders.

Just don't forget it.

I hope you enjoyed the lesson.

I welcome very much your questions and suggestions.

Coders' Guru

Life cycle of MQL4 program

Hi folks,

Today we are going to discuss a flustering issue for the most of new MQL4 comers; the life cycle of the MQL4 program.

What the first block of code is executed first and what the last block of code is executed, what's the phases of the MQL4 program. What's the different between the life cycle of the expert advisors and script? and a lot of questions about the MQL4 program life cycle.

Let's know first what happen when the MQL4 starts?

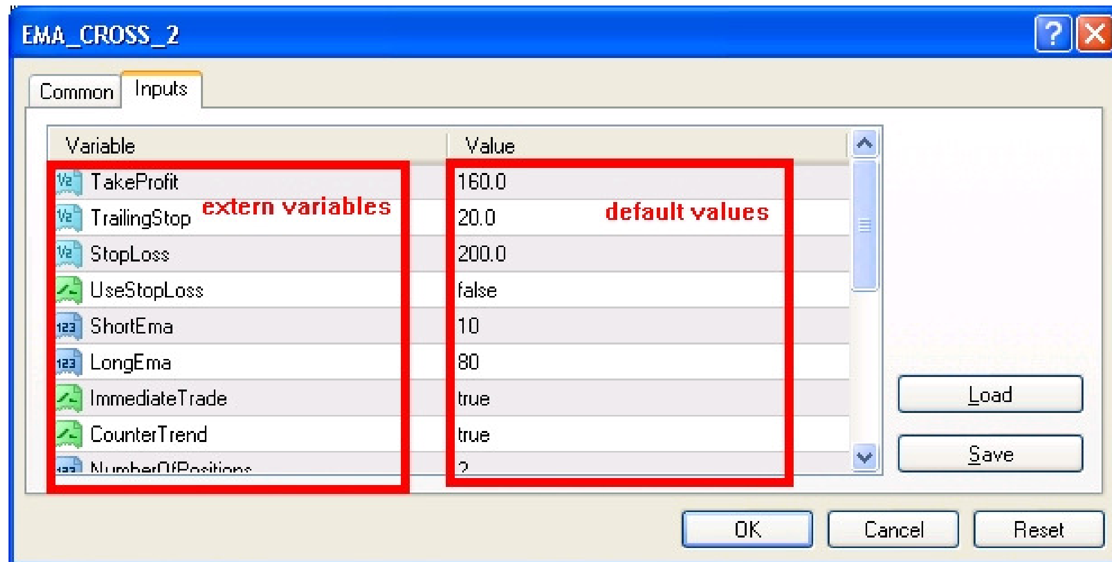
When the MQL4 program starts:

The first time you attach the MQL4 to chart the first thing MetaTrader will do is showing the input dialog depending on the type of the program:

Expert Advisors:

In this case you'll get the expert advisor input window (Figure 1) which enable you to set the parameters of the expert advisor if there was any (i.e. take profit and stop loss levels).

Each variable you declare using "extern" keyword will appear in the expert advisor input windows with its default value.



Indicators:

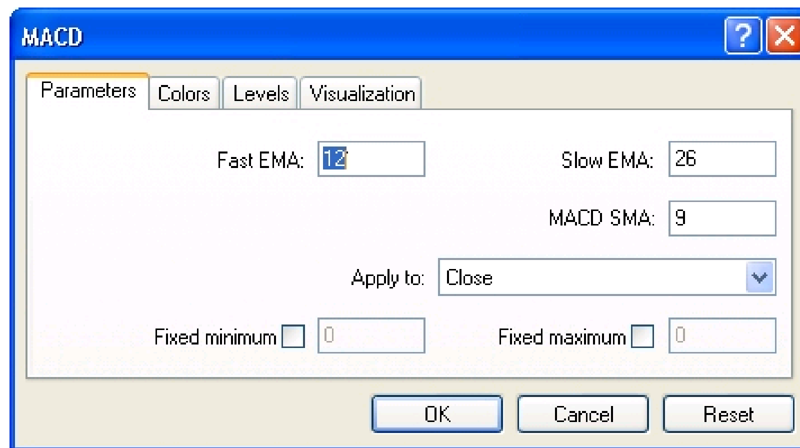
The same as expert advisors, the first time you attach an indicator to chart you'll get the parameters window (Figure 2) which enable you to set the parameters of the indicator, the colors of the indicator lines (Figure 3) and the levels of the sub-window chart (Figure 4).

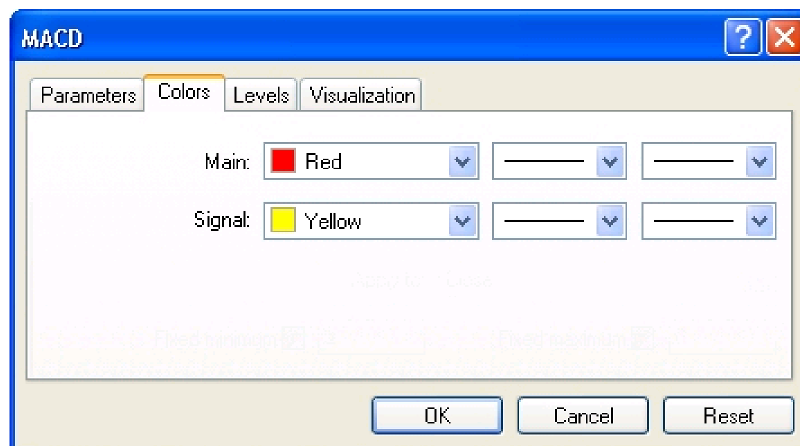
Each variable you declare using "extern" keyword will appear in the indicator parameters windows with its default value.

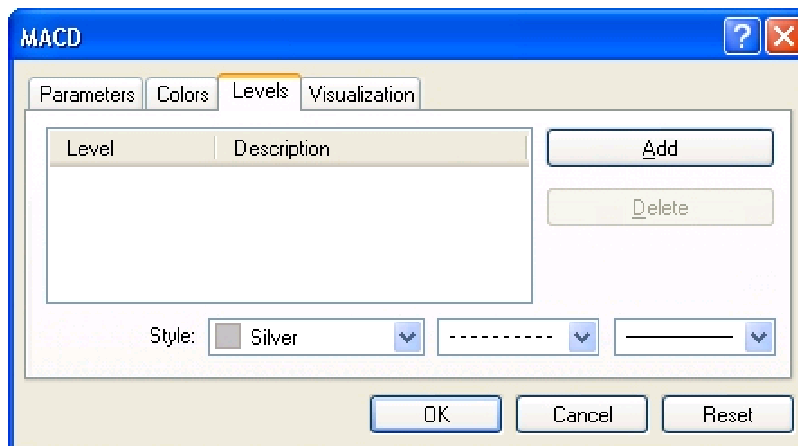
The colors are defined using the preprocessor keyword "#property indicator_colorN", for example:

#property indicator_color1 Silver
#property indicator_color2 Red

and the preset levels are defined using the preprocessor keywords "#property indicator_levelN", "#property indicator_levelcolor", "#property indicator_levelwidth" and "#property indicator_levelstyle".



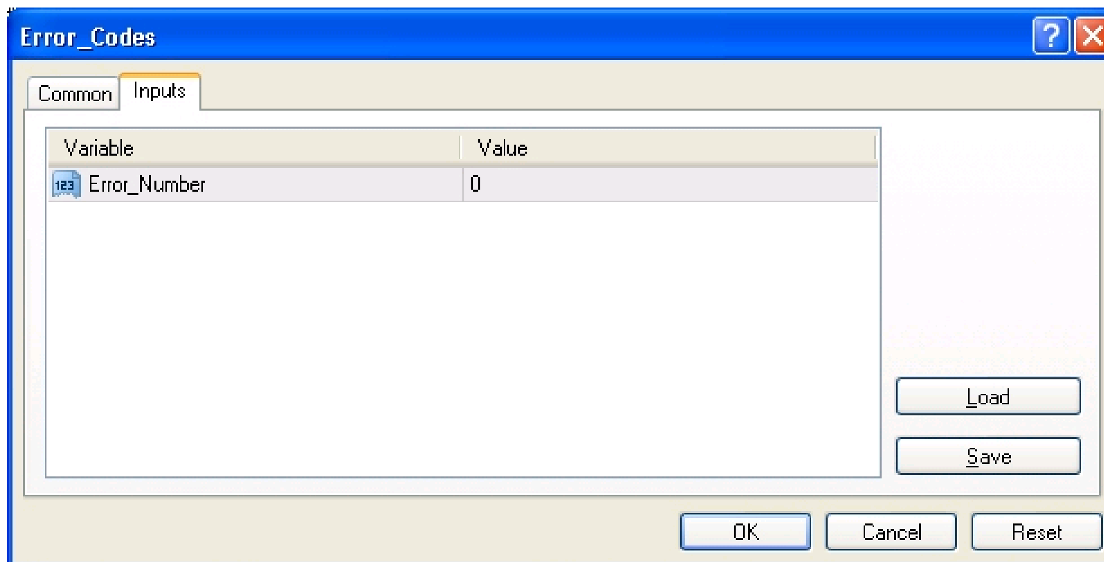




Scripts:

If there were any inputs coded in the script using the extern keyword, they will appear in the script input window (Figure 5) only if you used the preprocessor keyword "#property show_inputs".

If you used "#property show_confirm" preprocessor keyword a confirmation message box will appear before the script input window.



This is the first step MetaTrader will take in the journey of the MQL4 program life. Let's see what will happen nex?

Initialization phase?

The next phase in the MQL4 program cycle is the initialization phase.

Every line of code you wrote inside the block of init() function will be executed in this phase. This phase executed if one of these triggers occurred:

1- The first time you attach the program to a new chart (After the input window appearance).

2- When you start MetaTrader and there were opened chart(s) that hosts expert advisors or indicators. The script is executed only once (to the end of its life cycle) then de-initialized so MetaTrader wouldn't host opened scripts.

3- When you change the period of the chart and the program was hosted on one or more charts (except the scripts).

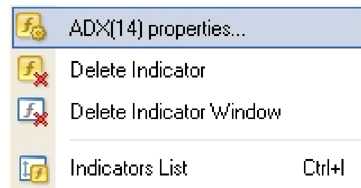
4- When you change the symbol of the chart and the program was hosted on one or more charts (except the scripts).

5- When you recompile the program in MetaEditor and the program was hosted on one or more charts (except the scripts).

6- When you change one of the program inputs (hitting F7 hot key to open expert advisor input window and right click the indicator then choose indicator properties from the context menu - Figure 6).

7- When you change the account the expert advisors will be initialized.

Unlike the above case, the initialization code is executed only one time during the life cycle of the program and will not be called again till the end of the program.



Execution phase:

After the initialization of the program it will wait for new quotation arrival from the server, every time MetaTrader receives new price quotation from the server it calls start() function and execute all the lines of the code in its block.

Note: The scripts is an exception because its start() function is executed once you attach it to the chart and immediately after the init() function then the script is detached from the chart.

The start() function have to finish its job first before receiving/reacting to new quotations, that means the code of start() function is executed line by line and only when MetaTrader find the return keyword it will be ready to call start() function again with the most new quotation.
If there were any quotations that arrived while the execution of start() function it would be ignored by the program,

Besides the new quotation arrival trigger, the start() function is executed in these cases:

1- For the indicators the start() function is recalled in the case of changing the chart symbol.

2- For the indicators the start() function is recalled in the case of changing the chart period

Note: The start() function will not be run when you open the expert advisor input window, you have to close it to enable MetaTrader to execute start() function again.

The execution phase is the longest phase in the program cycle life and in the most of programs the start() function block code is the most important code.

De-initialization phase?

Every program attached has its end, when the program ends its job the block of deinit() function will be executed.

These are the triggers of the deinit() function:

1- When you shut down MetaTader and the program was hosted on one or more charts.

2- When you close the chart that hosts the program.

3- Before the changing of the period of the chart that hosts the program.

4- Before the changing of the symbol of the chart that hosts the program.

5- When you recompile the program and the program was hosted on one or more charts.

6- When you change the inputs of the program.

7- When you change the account the expert advisors will be de-initialized.

These was the life cycle of the MQL4 program and it's the time to de-initialize the lesson.

Hope you enjoyed it,
Coders Guru

MarketInfo function

Hi folks,

We are going to complete our series of lessons that answer the question: "How to deal with currency pairs which are different than the current chart currency (The chart that hosts the expert advisor)? For example, how to make an expert advisor that can open Buy position of GBPUSD or USDCHF while the expert advisor is hosted on EURUSD chart?"

In the previous lesson we studied the Timeseries Access functions that enables us to access the price date of any chart (currency and timeframe) regardless of the chart that hosts the expert advisor. But they are not enough to write our expert advisor.

What's the Bid/Ask price we are going to use with OrderSend to open the order? We can't use the function Bid() and Ask() because they are returning the current currency Bid and Ask price.

Our magic function today is MarketInfo() function:

MarketInfo() function:

Syntax:

```
double MarketInfo( string symbol, int type)
```

Description:

The MarketInfo() function is the function you need to retrieve various market data of the given currency, for example: the current Bid price of the currency, the current Ask price of the currency, the Swap value of Buy or Selling the currency and the number of digits the currency uses.

Note: A part of this data are stored in the Predefined Variables but this data is exclusive for the current chart and MarketInfo() function give us more data about the currency.

This function can return the following important data:

MQL4 programs protection!

Hi folks!

I'm very fanatic for sharing MQL4 codes/ideas, however I've got a lot of requests asking me about MQL4 indicator/experts protection. I think it's not evil to use your programming talent to get some money if you don't want to share (it's a choice anyway).

Without protecting your MQL4 code it will be waste of time to try to commercialize your programs, that's why I'm writing this article.

Source code verse binary:

Just as a review and to emphasis that we are talking about how to protect the .ex4 files read the next section:

There are two kinds of files that you use with MetaTrader; the program files and the executable files:

The program files are normal text files but have one of these extensions: .mq4 and .mgh.
These files are the source code of the programs wrote in MQL4 programming language; source code means they could be opened for viewing or editing in MetaEditor.

The executable files are binary files (you can't open them for viewing or editing) and has the extension .ex4.
These files are the result of compiling the .mq4 files using MetaEditor, and these are the files that you can load them in MetaTrader and use them.

Compiling is an operation taken by a special program (called a compiler) to convert the program from text (readable) format to the binary format which the computer can understand (computers think in binary).
This is the general definition of compiling. Compiling in our case is converting the mq4 files to ex4 for file using MetaEditor program. We are doing this by opening the mq4 files in MetaEditor then pressing F5 hot key. MetaEditor will compile (convert) the file to ex4 format and keeping the mq4 file untouched MetaEditor will place the generated ex4 file at the same path as the mq4 file. We compile the mq4 files because the MetaTrader can't load any files except ex4 files.

The source code of MQL4 programs can't be protect because it's in text format and when you distribute it you intend to give the receiver the access to the source code of the program.
The executable version of the program is the only version that you can protect it. You protect it by write the protection code in the source code of the program then compile it to the ex4 format then distribute it to the user.

Kinds (ideas) of the protection:

We are going to discuss some ideas of MQL4 programs protection, maybe they are not the best but they are the common ideas, we are going to write some code to apply these ideas:

Password protection code:

This the widely used method to protect softwares in general and can be used to protect MQL4 programs. When the user buy your program you send him the program with a password and he can use the program without the password.

This is a simple code to apply password protection method:

```
int start()
  {
extern string Please_Enter_Password = "0";
// your code here....
int start()
{
   if (password != "viva metatrader") //change to the password you give the user!
   {
       Alert ("Wrong password!");
       return (0);
   }
// your code here....
}
```

In the code above the password is "viva metatrader" which is hard coded in the MQL4 file. You have to compile the program after adding this piece of code and send it to the user with the password.

Trial period protection:

If you want to give the user of the program a try-before-buy program you can limit the usage of your program by limited period of time and after this period the program will not work.
Use the code below to limit your program for period of time.

```
int start()
  {
   string expire_date = "2006.31.06"; //<-- hard coded datetime
   datetime e_d = StrToTime(expire_date);

   if (CurTime() >= e_d)
   {
      Alert ("The trial version has been expired!");
      return(0);
   }
   // your normal code!
   return(0);
  }
```

Limited account number protection:

In this method of protection you will ask the user to give you his account number and you write the following code to prevent the program to work with another accounts:

```
int start()
  {

  int hard_accnt = 11111; //<-- type the user account here before compiling
  int accnt = AccountNumber();

   if (accnt != hard_accnt)
   {
```

```
    Alert ("You can not use this account (" + DoubleToStr(accnt,0) + ") with this program!");
    return(0);
  }
  // your normal code!
  return(0);
 }
```

Limited account type protection:

In this method of protection you will allow the user to use the program in demo mode only and prevent him to use the program in live trading. It's not strong protection because the user can host the program in another instance of MetaTrader that runs in demo mode and trade manually in the real account instance.

```
int start()
 {

 bool demo_account = IsDemo();

  if (!demo_account)
  {
    Alert ("You can not use the program with a real account!");
    return(0);
  }
  // your normal code!
  return(0);
 }
```

DLL protection:

**This is the method of choice, write your own C++ DLL and let the MQL4 program export it.
In C++ you can do anything, for instance you can make the DLL communicated to your server (if you have one) enabling the user to download a certificate or even update the MQL4 program.**

I hope I can to write all purposes MQL4 protection DLL. I'll let you know when I write one.

Hope you enjoyed the article!

Object functions

Hi folks,

You use objects (Line studies, shapes, arrows and texts) as a visual aid to facilitate the analysis of chart. Drawing objects in MetaTrader manually is an easy task, you just have to choose the object you want to draw from the Insert menu or from the Line Studies toolbar (Figures 1 and 2).

Then you can modify/delete the created object by selecting the object on the chart and right click it then choose the operation you want from the context menu (Figure 3).

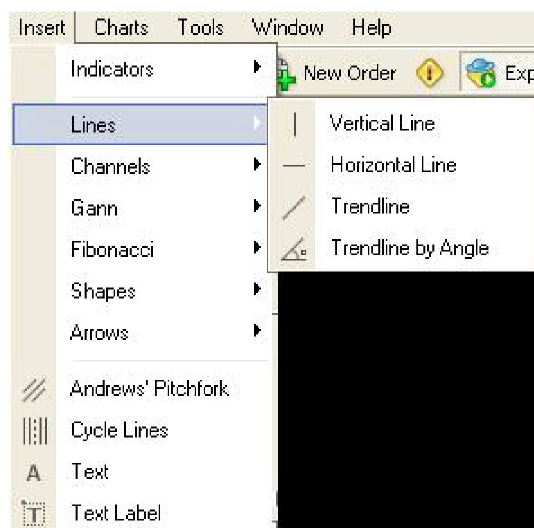


Figure 1 - Insert menu

Figure 2 - Line studies toolbar

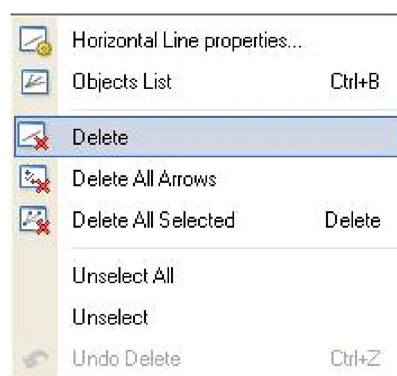


Figure 3 - Objects context menu

Today we are going to know how to programicaly work with objects in MQL4. We are going to study the Object functions which enable us to handle everything you can manually do with the objects and the things you can't do manually.
For example if you have hundred of objects on the chart and you want to delete them, it's a hard task to do it manually but in MQL4 it's a very easy task (as you'll see later)

Let's see the Object functions:

ObjectCreate:

Syntax:

```
bool ObjectCreate(string name, int type, int window, datetime time1, double price1, datetime time2=0,
double price2=0, datetime time3=0, double price3=0)
```

Description:

The *ObjectCreate* function creates a new object with the specified name and type, in the specified window and coordinates. The function returns true if the object successfully created otherwise it returns false. To know the exact error use GetLastError() function.

Object name:

The name of the object is the name you see in the Objects list window (Figure 4) and this name works as a handle for the object which you can use later to work with this object (For example to delete it or move it) so it must be unique.

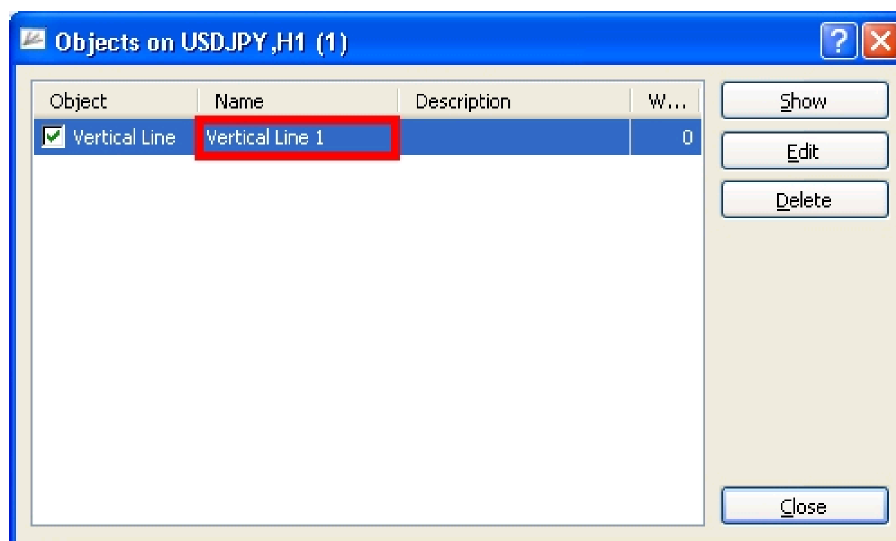


Figure 4 - Objects list window

Windows:

You can draw the object on the chart main window (The main window which show the price date) or on any of the chart sub-windows , you use the index of the window with the ObjectCreate function and you have to remember that:

1- The index of the chart main window is 0.
2- The chart sub-windows indexes start from 1 up to the number of the windows minus 1.
3- To get the number of window use the function WindowsTotal( ).
4- To find the index of a specific window by its indicator name use the function WindowFind(string name).

5- The index of the window must be less than the total of windows (retuned by WindowsTotal( ) function).

Coordinates:

You can use up to 3 coordinates for the object drawing depending on the type of the object, for example the Vertical line object uses only 1 coordinate, the Trend line object uses 2 coordinates and the Channel object use 3 coordinates.
The X (vertical) coordinate of the object is the time on the chart and the Y (horizontal) coordinate of the object is the price on the chart (Figure 5).
The first coordinate pairs (X and Y) are required parameters even if an object like the OBJ_LABEL will ignore them, while the second and third coordinates pairs are optional and you have to use them only with the object which use 2 or 3 coordinates.

Figure 5 - Coordinates

Parameters:

string name:

The string name of the object which you will use it later as a reference of object.

int type:

The type of the object you want to draw, you pass to the ObjectCreate function the integer representation of the object or the constant name of this integer, this is the table of the object integers representation and their constant names:

Constant Value Description

OBJ_VLINE	0	Vertical line. Uses time part of first coordinate.
OBJ_HLINE	1	Horizontal line. Uses price part of first coordinate.
OBJ_TREND	2	Trend line. Uses 2 coordinates.
OBJ_TRENDBYANGLE	3	Trend by angle. Uses 1 coordinate. To set angle of line use ObjectSet() function.
OBJ_REGRESSION	4	Regression. Uses time parts of first two coordinates.
OBJ_CHANNEL	5	Channel. Uses 3 coordinates.

OBJ_STDDEVCHANNEL	6	Standard deviation channel. Uses time parts of first two coordinates.
OBJ_GANNLINE	7	Gann line. Uses 2 coordinate, but price part of second coordinate ignored.
OBJ_GANNFAN	8	Gann fan. Uses 2 coordinate, but price part of second coordinate ignored.
OBJ_GANNGRID	9	Gann grid. Uses 2 coordinate, but price part of second coordinate ignored.
OBJ_FIBO	10	Fibonacci retracement. Uses 2 coordinates.
OBJ_FIBOTIMES	11	Fibonacci time zones. Uses 2 coordinates.
OBJ_FIBOFAN	12	Fibonacci fan. Uses 2 coordinates.
OBJ_FIBOARC	13	Fibonacci arcs. Uses 2 coordinates.
OBJ_EXPANSION	14	Fibonacci expansions. Uses 3 coordinates.
OBJ_FIBOCHANNEL	15	Fibonacci channel. Uses 3 coordinates.
OBJ_RECTANGLE	16	Rectangle. Uses 2 coordinates.
OBJ_TRIANGLE	17	Triangle. Uses 3 coordinates.
OBJ_ELLIPSE	18	Ellipse. Uses 2 coordinates.
OBJ_PITCHFORK	19	Andrews pitchfork. Uses 3 coordinates.
OBJ_CYCLES	20	Cycles. Uses 2 coordinates.
OBJ_TEXT	21	Text. Uses 1 coordinate.
OBJ_ARROW	22	Arrows. Uses 1 coordinate.
OBJ_LABEL	23	Text label. Uses 1 coordinate in pixels.

int window:

The index of the window you want to draw the object on, the chart main window has the index 0 and the sub chart starts from 1 to the total of the window - 1.

datetime time1:

The first vertical coordinate of the object. This parameter is required and you can set it to 0 if the object will not use the vertical coordinate.
This parameter is a datetime type (because the vertical coordinate of the chart is the time coordinate).

double price1:

The first horizontal coordinate of the object. This parameter is required and you can set it to 0 if the object will not use the horizontal coordinate.
This parameter is a double type (because the horizontal coordinate of the chart is the price coordinate). You have to note which currency pair you are going to draw on its chart because the prices ranges differs from currency to currency.

datetime time2:

The second vertical coordinate of the object. It's an optional parameter which you can use it if your object uses 2 coordinates.

double price2:

The second horizontal coordinate of the object. It's an optional parameter which you can use it if your object uses 2 coordinates.

datetime time3:

The third vertical coordinate of the object. It's an optional parameter which you can use it if your object uses 3 coordinates.

double price3:

The third horizontal coordinate of the object. It's an optional parameter which you can use it if your object uses 3 coordinates.

Example:

```
// new text object
if(!ObjectCreate("text_object", OBJ_TEXT, 0, D'2004.02.20 12:30', 1.0045))
{
Print("error: can't create text_object! code #",GetLastError());
return(0);
}
// new label object
if(!ObjectCreate("label_object", OBJ_LABEL, 0, 0, 0))
{
Print("error: can't create label_object! code #",GetLastError());
return(0);
}
ObjectSet("label_object", OBJPROP_XDISTANCE, 200);
ObjectSet("label_object", OBJPROP_YDISTANCE, 100);
```

ObjectDelete:

Syntax:

```
bool ObjectDelete(string name)
```

Description:

The *ObjectDelete* function deletes the object by its name (you have to provide the exact name of the object). If the object successfully deleted the function will return true, otherwise it will return false. Use GetLastError() function to get more details about the error.

Parameters:

string name:

The string name of the object you want to delete.

Example:

```
ObjectDelete("text_object");
```

ObjectDescription:

Syntax:

```
string ObjectDescription(string name)
```

Description:

The *ObjectDescription* function returns the description of the object (Figure 6).
For the objects OBJ_TEXT and OBJ_LABEL which has no description the returned value is the text drawn by these objects (Figure 7).

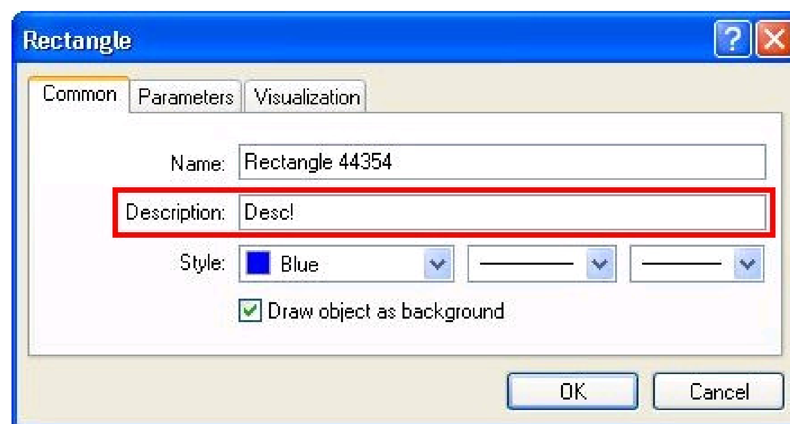


Figure 6- Object description

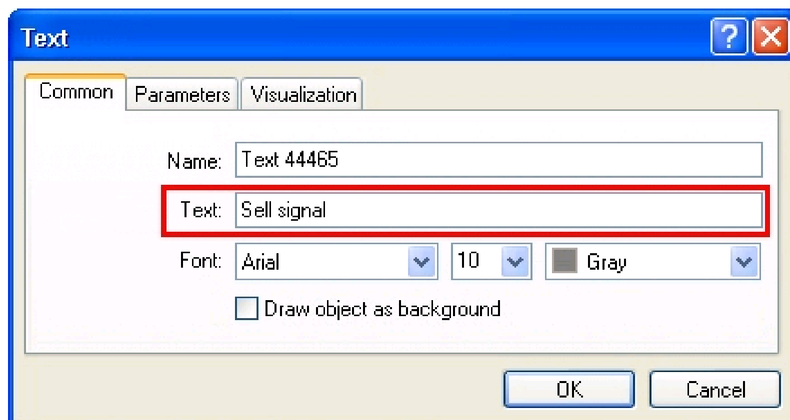


Figure 7

Parameters:

string name:

The string name of the object you want to get its description.

Example:

```
Print(ObjectDescription("Text 44465"));
```

ObjectFind:

Syntax:

```
int ObjectFind(string name)
```

Description:

The *ObjectFind* function searches all the drawn objects for the specified object name and returns the index of the window that host the object if found and returns -1 if the object not found. The chart main window is 0 index and the first chart sub-window is 1 then 2 etc.

Parameters:

string name:

The string name of the object you want to search for.

Example:

```
if(ObjectFind("line_object2")!=win_idx) return(0);
```

ObjectGet:

Syntax:

```
double ObjectGet(string name, int index)
```

Description:

The *ObjectGet* function returns the value of the specified object property of the given index.
For example to get the first vertical (time) of the object use ObjectGet(object_name, 0).

Parameters:

string name:

The string name of the object you want to get one of its proporties.

int index:

The property index which can be one of these values:

Constant Value Type Description

Constant	Value	Type	Description
OBJPROP_TIME1	0	datetime	Datetime value to set/get first coordinate time part.
OBJPROP_PRICE1	1	double	Double value to set/get first coordinate price part.
OBJPROP_TIME2	2	datetime	Datetime value to set/get second coordinate time part.
OBJPROP_PRICE2	3	double	Double value to set/get second coordinate price part.
OBJPROP_TIME3	4	datetime	Datetime value to set/get third coordinate time part.
OBJPROP_PRICE3	5	double	Double value to set/get third coordinate price part.
OBJPROP_COLOR	6	color	Color value to set/get object color.
OBJPROP_STYLE	7	int	Value is one of STYLE_SOLID, STYLE_DASH, STYLE_DOT, STYLE_DASHDOT, STYLE_DASHDOTDOT constants to set/get object line style.
OBJPROP_WIDTH	8	int	Integer value to set/get object line width. Can be from 1 to 5.
OBJPROP_BACK	9	bool	Boolean value to set/get background drawing flag for object.
OBJPROP_RAY	10	bool	Boolean value to set/get ray flag of object.
OBJPROP_ELLIPSE	11	bool	Boolean value to set/get ellipse flag for fibo arcs.
OBJPROP_SCALE	12	double	Double value to set/get scale object property.
OBJPROP_ANGLE	13	double	Double value to set/get angle object property in degrees.
OBJPROP_ARROWCODE	14	int	Integer value or arrow enumeration to set/get arrow code object property.
OBJPROP_TIMEFRAMES	15	int	Value can be one or combination (bitwise addition) of object visibility constants to set/get timeframe object property.
OBJPROP_DEVIATION	16	double	Double value to set/get deviation property for Standard deviation objects.
OBJPROP_FONTSIZE	100	int	Integer value to set/get font size for text objects.
OBJPROP_CORNER	101	int	Integer value to set/get anchor corner property for label objects. Must be from 0-3.
OBJPROP_XDISTANCE	102	int	Integer value to set/get anchor X distance object property in pixels.
OBJPROP_YDISTANCE	103	int	Integer value is to set/get anchor Y distance object property in pixels.
OBJPROP_FIBOLEVELS	200	int	Integer value to set/get Fibonacci object level count. Can be from 0 to 32.
OBJPROP_LEVELCOLOR	201	color	Color value to set/get object level line color.
OBJPROP_LEVELSTYLE	202	int	Value is one of STYLE_SOLID, STYLE_DASH, STYLE_DOT, STYLE_DASHDOT, STYLE_DASHDOTDOT constants to set/get object level line style.
OBJPROP_LEVELWIDTH	203	int	Integer value to set/get object level line width. Can be from 1 to 5.
OBJPROP_FIRSTLEVEL+*n*	210+n	int	Fibonacci object level index, where *n* is level index to set/get. Can be from 0 to 31.

Example:

```
color oldColor=ObjectGet("hline12", OBJPROP_COLOR);
```

ObjectGetFiboDescription:

Syntax:

```
string ObjectGetFiboDescription(string name, int index)
```

Description:

The *ObjectGetFiboDescription* function returns the description of the given Fibonacci object level. For example in Figure 8 the description of the second line from bottom (its index is 0) is "23.6". The index of Fibonacci object levels start from bottom upward and you can use up to 32 levels.

Figure 8

Parameters:

string name:

The string name of the Fibonacci object you want to get its level description.

int index:

The index of the level (line) you want to get its level description.

Example:

```
Print(ObjectGetFiboDescription("Fibo 2638", 1));
```

ObjectGetShiftByValue:

Syntax:

```
int ObjectGetShiftByValue(string name, double value)
```

Description:

The *ObjectGetShiftByValue* function returns the index of the bar (shifted from the current - 0 upward) for the given price of the given object. This price is calculated with the first and second coordinates of the object (using liner equation).
Foe example in figure 9 the bar index for the price 1.2156 of the Trendline object is the bar 11.

Figure 9

Parameters:

string name:

The string name of the object you want to get the bar index for one of its prices (values)

double value:

The price value of the object.

Example:

```
Print(ObjectGetShiftByValue("Trendline 5879", 1.2156));
```

In the next lesson we are going to continue with the remaining Object Function and we are going to make a simple MQL program to apply what we learnt about the objects programming.

Hope you find the lesson helpful one and hope to see your comments!

Coders Guru!

Object Functions - Part 2

Hi folks,

Today we are going to continue with the remaining of Object Functions and we have a simple MQL program to apply some of what we have learnt about drawing objects programicaly.

Let's see what left of the object functions:

ObjectMove:

Syntax:

```
bool ObjectMove(string name, int point, datetime time1, double price1)
```

Description:

The *ObjectMove* function changes the specified coordinate of the object to new x and y coordinate.
Any object has up to 3 coordinates and they are indexed 0, 1 and 2. You specify the coordinate you want to change with its index (point parameter) then set new x and y values for the new coordinate (time1 and time2).
The function returns true if it successfully had changed the coordinate and false otherwise.

Note: If you have a 2 or 3 coordinates object and want to change the 2 or 3 coordinates, you have to use ObjectMove function 2 or 3 times for every coordinate.

Parameters:

string name:

The string name of the object you want to change its coordinate.

int point:

Coordinate index (0, 1, 2) you want to change it.

datetime time1:

The first vertical coordinate of the object. This parameter is a datetime type (because the vertical coordinate of the chart is the time coordinate).

double price1:

The first horizontal coordinate of the object. This parameter is a double type (because the horizontal coordinate of the chart is the price coordinate).

Note: You have to note which currency pair you are going to draw on its chart because the prices ranges differs from currency to currency.

Example:

```
ObjectMove("MyTrend", 1, D'2005.02.25 12:30', 1.2345);
```

ObjectName:

Syntax:

```
string ObjectName(int index)
```

Description:

The *ObjectName* function returns the object name for the given index from the object list. To check the error use GetLastError() function.

Parameters:

int index:

The index of the object you want to get its name.

Note: The index must be equal to 0 or greater, and be less than the total of the objects count returned by ObjectsTotal() function.

Example:

```
int obj_total=ObjectsTotal();
string name;
for(int i=0;i<obj_total;i++)
{
name=ObjectName(i);
Print(i,"Object name is " + name);
}
```

OrderSend - Type of orders!

Hi folks,

"OK, now I have a brain pain! I just spent the morning scanning thru the MQL4 and MetaTrader help file pages to try and track down if I missed anything."

Could you please answer me only one question:

When I open a Buy order using OrderSend function, Should I use the Ask or the Bid price? What about the Stop loss is it going below or above the Ask or the Bid price? What about the Take profit?

When I open a Sell order using OrderSend function, Should I use the Ask or the Bid price? What about the Stop loss is it going below or above the Ask or the Bid price? What about the Take profit?

What about BUYLIMIT, BUYSTOP, SELLLIMIT and SELLSTOP orders?

Is that the only one question? Well here's the answer!

OP_BUY

Ask
Ask-StopLoss
Ask+TakeProfit

You Buy at the current Ask price of the currency!
You set the StopLoss Below (-) the Ask price!
You set the TakeProfit Above (+) the Ask price!

Example:

OrderSend(Symbol(),OP_BUY,Lots,Ask,slippage,
Ask-StopLoss*Point,Ask+TakeProfit*Point,"comment",0,0,Green);

OP_SELL

Bid
Bid+StopLoss
Bid-TakeProfit

You Sell at the current Bid price of the currency!
You set the StopLoss Above (+) the Bid price!
You set the TakeProfit Below (-) the Bid price!

Example:

OrderSend(Symbol(),OP_SELL,Lots,Bid,slippage,
Bid+StopLoss*Point,Bid-TakeProfit*Point, "comment",0,0,Green);

OP_BUYLIMIT

Ask-Level
Ask-Level-StopLoss
Ask-Level +TakeProfit

You Buy at future price level Below the current Ask price of the currency!
You set the StopLoss Below (-) the Ask - the level price!
You set the TakeProfit Above (+) the Ask - the level price!

Example:

OrderSend(Symbol(),OP_BUYLIMIT,Lots,Ask-Level*Point,slippage,
(Ask-Level*Point)-StopLoss*Point,
(Ask-Level*Point)+TakeProfit*Point,"comment",0,0,Green);

OP_BUYSTOP

Ask+Level
Ask+Level -StopLoss
Ask+Level +TakeProfit

You Buy at future price level Above the current Ask price of the currency!
You set the StopLoss Below (-) the Ask + the level price!
You set the TakeProfit Above (+) the Ask + the level price!

Example:

OrderSend(Symbol(),OP_BUYSTOP,Lots,Ask+Level*Point,slippage,
(Ask+Level*Point)-StopLoss*Point,
(Ask+Level*Point)+TakeProfit*Point,"comment",0,0,Green);

OP_SELLLIMIT

Bid+Level
Bid +Level +StopLoss
Bid+Level –TakeProfit

You Sell at future price level Above the current Bid price of the currency!
You set the StopLoss Above (+) the Bid + the level price!
You set the TakeProfit Below (-) the Bid + the level price!

Example:

OrderSend(Symbol(),OP_SELLLIMIT,Lots,Bid+Level*Point,slippage,
(Bid+Level*Point)+StopLoss*Point,
(Bid +Level*Point)-TakeProfit*Point,"comment",0,0,Green);

OP_SELLSTOP

Bid-Level
Bid -Level +StopLoss
Bid-Level-TakeProfit

You Sell at future price level Below the current Bid price of the currency!
You set the StopLoss Above (+) the Bid + the level price!
You set the TakeProfit Below (-) the Bid - the level price!

Example:

OrderSend(Symbol(),OP_SELLSTOP,Lots,Bid-Level*Point,slippage,
(Bid-Level*Point)+StopLoss*Point,
(Bid -Level*Point)-TakeProfit*Point,"comment",0,0,Green);

Predefined variables

Hi folks,

**Today we are going to talk about a very important topic, the Predefined variables.
I still considering them functions (I'm going to tell you why later) but I'll use Predefined variables as MetaQuotes has mentioned them.**

What are the Predefined variables?

The Predefined variables are set of the most important variables which MetaTrader set continuously for every loaded chart. They are price related variables that reflect the current price data the chart had got.

These is the list of the predefined variables:

Ask, Bid, Bars, Close, Open, High, Low, Time, Digits, Point and Volume

I'm calling them functions because:

**Ask, Bid, Bars, Close, Open, High, Low, Time, Digits, Point and Volume are functions Although MetaQuotes called them predefined variables.
Variable means "a space in memory and data type you specify", while 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.**

Anyway, it doesn't matter what should you call them, the really matter is how to use them!

Let's study the Predefined variables set:

StringConcatenate:

Syntax:

```
double Ask
```

Description:

The *Ask* function return the last known Ask price (the sell price) for the current symbol. To be sure it reflects the actual market price you have to use the RefreshRates() function.

Note: To get the Ask price for a

Parameters:

***Example*:**

Questions - Close price

How can i get the close, low, high price of current bar? When I use Close[0], iClose(Symbol(), 0, 0) to get close price of current bar, thess two values are always the same as Open[0](when running tester tool of MetaTrader, right now I'm running MIG Trade Station, demo account) ? The same for low and high prices? Please tell me how can i figure out it. Many thanks for reading!

---------- REPLY -----------------

Hi there!

Close[0] returns the close price of the current bar (The same is true for the High[0] return the high price of the current bar, Low[0] returns the low price of the current bar and Open[0] returns the open price of the current bar)

iClose function is very like Close function but it can return the close price for another symbol and timframe. If you used it like this:

iClose(NULL,0,0)

it will equal to Close[0]. (The same is true for the iHigh, iLow and iOpen functions).

Coders Guru

String functions

Hi folks,

Our set of functions today are the String Functions. They are the MQL4 functions we need to handle the string variables!
Let's give the string data type an overview first!

string data type:

The string data type is an array of characters enclosed in double quote (").
This array of characters is an array which holds one character after another, starting at index 0. After the last character of data, a NULL character is placed in the next array location. It does not matter if there are unused array locations after that.
A NULL character is a special character (represented by the ASCII code 0) used to mark the end of this type of string.

See figure 1 for a simple representation of the string constant "hello" in the characters array.

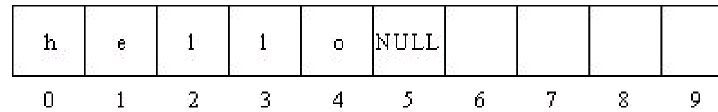


Figure 1 – Characters array

MQL4 limits the size of the string variable to 255 characters and any character above 255 characters will generate this error: (too long string (255 characters maximum)).

We use the keyword string to create a string variable.

For example:

```
string str1 = "Hello world!, with you coders guru";
string str2 = "Copyright © 2005, \"Forex-tsd forum\"."; //Notice the use of (") character.
string str3 = "1234567890";
```

Now let's study the MQL4 functions concerned about the Strings:

StringConcatenate:

Syntax:

```
string StringConcatenate( ... )
```

Description:

The *StringConcatenate* function take the set of parameters passed to it and glue them together and returns them as a string.

Note: (...) means that you can pass to this function any kind of data type and any number of parameters separated by comma.

You can pass to this function any kind of data type except the arrays and any number of parameters separated by comma very like the Print(), Alert() and Comment() functions.
If you want to pass a double data type to this function the number of digits will be used are 4 digits if you want to change this number you have to use the function DoubleToStr().
The bool, color and datatime passed to function will be printed in their numeric presentation. , if you want to print them in string format use the string name of the bool or the color value and for the datetime use the function TimeToStr().

Note: You can use the + operator to add string to another string for example "coders" + "guru" will produce "codersguru", however MetaTrader saying that StringConcatenate() function works faster gluing strings together.

Parameters:

This function takes any number of parameters:

...

Any values, separated by commas.

Example:

```
string text;
text=StringConcatenate("Account free margin is ", AccountFreeMargin(), "Current time is ",
TimeToStr(CurTime()));
// slow text="Account free margin is " + AccountFreeMargin() + "Current time is " + TimeToStr(CurTime())
Print(text);
```

StringFind:

Syntax:

```
int StringFind( string text, string matched_text, int start=0)
```

Description:

The *StringFind* function search for substring (part of the given string for example "gur" is substring from the string "codersguru") in a given string, if the function find the substring it returns its position, if the function didn't find the substring it returns -1.

Parameters:

This function takes two parameters:

string text

The string you want to search in it for the substring.

string matched_text

The substring you want to find it in the string.

int start=0

The index of the starting position for the search. the first character in the string has the index 0 then 1 then 2 etc.
The default value is 0 which means to search from the start of the string.

Example:

```
string text="The quick brown dog jumps over the lazy fox";
int index=StringFind(text, "dog jumps", 0);
if(index!=16)
Print("oops!");
```

StringGetChar:

Syntax:

```
int StringGetChar( string text, int pos)
```

Description:

The *StringGetChar* function returns the character in a given position from a given string. it return it as character code (the ASCII code of the character).

Parameters:

This function takes two parameters:

string text

The string you want to get the character from.

int pos

The position of the character you want to get, don't forget that the string array starts with 0.

Example:

```
int char_code=StringGetChar("abcdefgh", 3);
// char code 'c' is 99
```

StringLen:

Syntax:

```
int StringLen( string text)
```

Description:

The *StringLen* function returns how many character (the length) of the given string.

Parameters:

This function takes only one parameter:

string text

The string you want to get its length..

Example:

```
string str="some text";
if(StringLen(str)<5) return(0);
```

StringSetChar:

Syntax:

```
string StringSetChar( string text, int pos, int value)
```

Description:

The *StringSetChar* function change one character of the given string in a given position to another character and returns the new string.

Parameters:

This function takes three parameters:

string text

The string you want change a character in it.

int pos

The position of the character you want to change.

int value

The new character you want to set it.

Example:

```
string str="abcdefgh";
string str1=StringSetChar(str, 3, 'D');
// str1 is "abcDefgh"
```

StringSubstr:

Syntax:

```
string StringSubstr( string text, int start, int count=EMPTY)
```

Description:

The *StringSubstr* function extracts the string from given position to a given count (ex: "codersguru" the position is 3 and the count is 2 then the extracted string is "de"). The function returns the extracted string if any or it returns EMPTY string.

Parameters:

This function takes three parameters:

string text

The string you want to extract from it.

int start

Where to start your extracting.

int count=EMPTY

The count of character you want to extract. The default value is EMPTY.

Example:

```
string text="The quick brown dog jumps over the lazy fox";
string substr=StringSubstr(text, 4, 5);
// subtracted string is "quick" word
```

Timeseries access functions - Part 1

Hi folks,

I've got a lot of messages asking me "How to deal with currency pairs which are different than the current chart currency (The chart that hosts the expert advisor)? For example, how to make an expert advisor that can open Buy position of GBPUSD or USDCHF while the expert advisor is hosted on EURUSD chart?" Good question, Huh?

**To reply these related questions we have to start with studding the Timeseries Access functions.
The Timeseries Access functions are a set of functions that enables us to access the price date of any chart (currency and timeframe) regardless of the chart that hosts the expert advisor.**

Let's give these functions a brief look:

iBars

Syntax:

```
int iBars( string symbol, int timeframe)
```

Description:

The *iBars* function returns the number of Bars of the specified chart, you specify the chart by the symbol name and the timeframe.

Note: If the chart of the symbol and timeframe you are trying to get its price data is not opened, MetaTrader will try to connect to the server to retrieve the request price data. In this case and with the all of the Timeseries access functions you have to check the the last was the error # 4066 - ERR_HISTORY_WILL_UPDATED (which means the requested data is under updating) to be sure that the price data is up-to-date. And retry your request for the price data.

Note: If you want to get the number of bars of the current chart (symbol and timeframe) use Bars function.

Parameters:

string symbol:

The symbol (currency pair) of the chart you want to get its bars number.
Use NULL if you want to use current symbol.

int timeframe:

The timeframe (in integers, ex: 1,5,30,60 etc) of the chart you want to get its bars number.
Use 0 if you want to use current timeframe.

Note: You can use the integer representation of the timeframe (period in minutes) or you can use the timeframes constants:

PERIOD_M1	1	1 minute.
PERIOD_M5	5	5 minutes.
PERIOD_M15	15	15 minutes.
PERIOD_M30	30	30 minutes.
PERIOD_H1	60	1 hour.
PERIOD_H4	240	4 hour.
PERIOD_D1	1440	Daily.
PERIOD_W1	10080	Weekly.
PERIOD_MN1	43200	Monthly.
0 (zero)	0	Timeframe used on the chart.

Example:

```
Print("Bar count on the 'EUROUSD' symbol with PERIOD_H1 is",iBars("EUROUSD",PERIOD_H1));
```

iBarShift:

Syntax:

```
int iBarShift( string symbol, int timeframe, datetime time, bool exact=false)
```

Description:

The *iBarShift* function takes the open time of the bar for a specified symbol and timeframe and searches for this bar and returns it if found otherwise it returns -1.
If you want the *iBarShift* function to return the nearest bar to the given open time set the exact parameter to true.

Parameters:

string symbol:

The symbol (currency pair) of the chart you want to get its bars number.
Use NULL if you want to use current symbol.

int timeframe:

The timeframe (in integers, ex: 1,5,30,60 etc) of the chart you want to get its bars number.
Use 0 if you want to use current timeframe.

datetime time:

The open time of the bar you want to search for

bool exact:

The search mode, use exact =false (default) and and the function will return -1 if the open time not found.
And use exact =true and the function will return the nearest bar to the given open time.

Example:

```
datetime some_time=D'2004.03.21 12:00';
int shift=iBarShift("EUROUSD",PERIOD_M1,some_time);
Print("shift of bar with open time ",TimeToStr(some_time)," is ",shift);
```

iClose:

Syntax:

```
double iClose( string symbol, int timeframe, int shift)
```

Description:

The *iClose* function returns the Close price for the given bar (the shift parameter is the bar number) of the given symbol and timeframe. The function returns 0 if the history data not loaded.

Note: It's very important (and always error's source) to know that the current bar is 0 and the previous bar is 1 etc.

Note: If you want to get the Close price (of a bar) of the current chart (symbol and timeframe) use Close[int shift] function.

Parameters:

string symbol:

The symbol (currency pair) of the chart you want to get its bars number.
Use NULL if you want to use current symbol.

int timeframe:

The timeframe (in integers, ex: 1,5,30,60 etc) of the chart you want to get its bars number.
Use 0 if you want to use current timeframe.

int shift:

The index of the bar you want to get its Close price.

Example:

```
Print("Current Close price for USDCHF H1: ",iClose("USDCHF",PERIOD_H1,i));
```

iHigh:

Syntax:

```
double iHigh( string symbol, int timeframe, int shift)
```

Description:

The *iHigh* function returns the High price for the given bar (the shift parameter is the bar number) of the given symbol and timeframe. The function returns 0 if the history data not loaded.

Note: If you want to get the High price (of a bar) of the current chart (symbol and timeframe) use High[int shift] function.

Parameters:

string symbol:

The symbol (currency pair) of the chart you want to get its bars number.
Use NULL if you want to use current symbol.

int timeframe:

The timeframe (in integers, ex: 1,5,30,60 etc) of the chart you want to get its bars number.
Use 0 if you want to use current timeframe.

int shift:

The index of the bar you want to get its High price.

Example:

```
Print("Current High price for USDCHF H1: ",iHigh("USDCHF",PERIOD_H1,i));
```

iLow:

Syntax:

```
double iLow( string symbol, int timeframe, int shift)
```

Description:

The *iLow* function returns the Low price for the given bar (the shift parameter is the bar number) of the given symbol and timeframe. The function returns 0 if the history data not loaded.

Note: If you want to get the Low price (of a bar) of the current chart (symbol and timeframe) use Low[int shift] function.

Parameters:

string symbol:

The symbol (currency pair) of the chart you want to get its bars number.
Use NULL if you want to use current symbol.

int timeframe:

The timeframe (in integers, ex: 1,5,30,60 etc) of the chart you want to get its bars number.
Use 0 if you want to use current timeframe.

int shift:

The index of the bar you want to get its Low price.

Example:

```
Print("Current Low price for USDCHF H1: ",iLow("USDCHF",PERIOD_H1,i));
```

iOpen:

Syntax:

```
double iOpen( string symbol, int timeframe, int shift)
```

Description:

The *iOpen* function returns the Open price for the given bar (the shift parameter is the bar number) of the given symbol and timeframe. The function returns 0 if the history data not loaded.

Note: If you want to get the Open price (of a bar) of the current chart (symbol and timeframe) use Open[int shift] function.

Parameters:

string symbol:

The symbol (currency pair) of the chart you want to get its bars number.
Use NULL if you want to use current symbol.

int timeframe:

The timeframe (in integers, ex: 1,5,30,60 etc) of the chart you want to get its bars number.
Use 0 if you want to use current timeframe.

int shift:

The index of the bar you want to get its Open price.

Example:

```
Print("Current Open price for USDCHF H1: ",iOpen("USDCHF",PERIOD_H1,i));
```

iTime:

Syntax:

```
datetime iTime( string symbol, int timeframe, int shift)
```

Description:

The *iTime* function returns the Open Time of the given bar (the shift parameter is the bar number) of the given symbol and timeframe.

Note: If you want to get the bar Open Time (of a bar) of the current chart (symbol and timeframe) use Time[int shift] function.

Parameters:

string symbol:

The symbol (currency pair) of the chart you want to get its bars number.
Use NULL if you want to use current symbol.

int timeframe:

The timeframe (in integers, ex: 1,5,30,60 etc) of the chart you want to get its bars number.
Use 0 if you want to use current timeframe.

int shift:

The index of the bar you want to get its Open Time.

Example:

```
Print("The Open Time of the current bar for USDCHF H1: ",iTime("USDCHF",PERIOD_H1,i));
```

iVolume:

Syntax:

```
double iVolume( string symbol, int timeframe, int shift)
```

Description:

The *iVolume* function returns the Tick Volume value of the given bar (the shift parameter is the bar number) of the given symbol and timeframe.

Note: Volume is simply the number of shares (or contracts) traded during a specified time frame (e.g., hour, day, week, month, etc).

Note: In MQL if you want to check if the tick is the first tick of the new bar you can check Volume[0] it will be 0 if it's the first tickof the new bar. Example:

```
//---- go trading only for first tiks of new bar
if(Volume[0]>1) return;
```

Note: If you want to get the Tick Volume value (of a bar) of the current chart (symbol and timeframe) use Volume[int shift] function.

Parameters:

string symbol:

**The symbol (currency pair) of the chart you want to get its bars number.
Use NULL if you want to use current symbol.**

int timeframe:

**The timeframe (in integers, ex: 1,5,30,60 etc) of the chart you want to get its bars number.
Use 0 if you want to use current timeframe.**

int shift:

The index of the bar you want to get its Tick Volume value.

Example:

```
Print("Current Tick Volume for USDCHF H1: ",iVolume("USDCHF",PERIOD_H1,i));
```

Highest:

Syntax:

```
int Highest( string symbol, int timeframe, int type, int count=WHOLE_ARRAY, int start=0)
```

Description:

The *Highest* function calculate the Highest value of the specified type (Close price, Open price, High price etc) for a specified bars of the given symbol and timeframe.

You use the type parameter to determine the type of the values to be calculated and the parameters count and start determine the bars to be calculated.

Parameters:

string symbol:

The symbol (currency pair) of the chart you want to get its bars number.
Use NULL if you want to use current symbol.

int timeframe:

The timeframe (in integers, ex: 1,5,30,60 etc) of the chart you want to get its bars number.
Use 0 if you want to use current timeframe.

int type:

The type of values (Series array) to be calculated, it can be one of these types:

Constant Value Description
MODE_OPEN 0 Open price.
MODE_LOW 1 Low price.
MODE_HIGH 2 High price.
MODE_CLOSE 3 Close price.
MODE_VOLUME 4 Volume.
MODE_TIME 5 Bar Open time.

int count:

How many bars you want to calculate its Highest value, use this parameter with start parameter to determine the range of bars to be calculated. The default value is WHOLE_ARRAY which means all the bars (values in the series array).

int start:

The bar to start the calculation from, the default value is 0 which means to start from the current bar.

Example:

```
double val;
// calculating the highest value in the range from 5 element to 25 element
// indicator charts symbol and indicator charts time frame
val=High[Highest(NULL,0,MODE_HIGH,20,4)];
```

Lowest:

Syntax:

```
int Lowest( string symbol, int timeframe, int type, int count=WHOLE_ARRAY, int start=0)
```

Description:

The *Lowest* function calculate the Lowest value of the specified type (Close price, Open price, High price etc) for a specified bars of the given symbol and timeframe.

You use the type parameter to determine the type of the values to be calculated and the parameters count and start determine the bars to be calculated.

Parameters:

string symbol:

The symbol (currency pair) of the chart you want to get its bars number.
Use NULL if you want to use current symbol.

int timeframe:

The timeframe (in integers, ex: 1,5,30,60 etc) of the chart you want to get its bars number.
Use 0 if you want to use current timeframe.

int type:

The type of values (Series array) to be calculated.

int count:

How many bars you want to calculate its Highest value, use this parameter with start parameter to determine the range of bars to be calculated. The default value is WHOLE_ARRAY which means all the bars (values in the series array).

int start:

The bar to start the calculation from, the default value is 0 which means to start from the current bar.

Example:

```
double val=Low[Lowest(NULL,0,MODE_LOW,10,10)];
```

Now we have the tools to access the price data of the other charts, the question is: Is this enough to write our expert advisor that deals with other currency pairs? No!

In the coming lesson we are going to know the most important function needed to write this kind of expert advisors; the MarketInfo() function.

I hope you find it a useful lesson!

Windows functions

Hi folks,

We are going to study today a set of functions that responsible of handling the chart windows. These functions are infrequently used (expect WindowsTotal(), WindowFind() and ScreenShot functions) but as a MQL4 programmer you have to know them.

These are the Windows functions set:

BarsPerWindow:

Syntax:

```
int BarsPerWindow( )
```

Description:

The BarsPerWindow function returns the number of bars visible for the user on the chart.

Note: Resizing the chart window or changing the chart period changes this number.

Parameters:

The function doesn't take any parameters.

Example:

```
// work with visible bars.
int bars_count=BarsPerWindow();
int bar=FirstVisibleBar();
for(int i=0; i<bars_count; i++,bar--)
{
// ...
}
```

```
caption=FirstVisibleBar()
type=int

Function returns index of the first visible bar.

// work with visible bars.
int bars_count=BarsPerWindow();
int bar=FirstVisibleBar();
for(int i=0; i<bars_count; i++,bar--)
{
// ...
}
```

PriceOnDropped:

Syntax:

```
double PriceOnDropped( )
```

Description:

The *PriceOnDropped* function returns the price of the currency of the chart from the point you dragged and release the expert advisor or the script at. When you click the expert advisor or the script in the navigator window and hold the left button mouse then drag it and release the left mouse buttons at any point on the chart; the *PriceOnDropped* returns the price at this point.

This function works only with the expert advisors and script and doesn't work with the custom indicators.

Parameters:

The function doesn't take any parameters.

Example:

```
double drop_price=PriceOnDropped();
datetime drop_time=TimeOnDropped();
//---- may be undefined (zero)
if(drop_time>0)
{
ObjectCreate("Dropped price line", OBJ_HLINE, 0, drop_price);
ObjectCreate("Dropped time line", OBJ_VLINE, 0, drop_time);
}
```

TimeOnDropped:

Syntax:

datetime TimeOnDropped( )

Description:

The *TimeOnDropped* function returns the time of on the chart where you dragged and release the expert advisor or the script at. This function too works only with the expert advisors and script and doesn't work with the custom indicators.

Parameters:

The function doesn't take any parameters.

Example:

```
double drop_price=PriceOnDropped();
datetime drop_time=TimeOnDropped();
//---- may be undefined (zero)
if(drop_time>0)
{
ObjectCreate("Dropped price line", OBJ_HLINE, 0, drop_price);
ObjectCreate("Dropped time line", OBJ_VLINE, 0, drop_time);
}
```

ScreenShot:

Syntax:

bool ScreenShot(**string** filename, **int** size_x, **int** size_y, **int** start_bar=-1, **int** chart_scale=-1, **int** chart_mode=-1)

Description:

The *ScreenShot* function save a screen shot of the current chart. It saves the screen shot as GIF file in one of two folder:
If you used the function in the live mode it'll save the screen shot in terminal_dir\experts\files and its subdirectories. If you used the function in the testing mode it'll save the screen shot in terminal_dir\tester\files and its subdirectories.

The function returns True if it successfully saved the screen shot and false if it failed.

Parameters:

The function takes 6 parameters.

string filename:

The file name that the function will save the screen shot to. You can combine the file name with a subdirectory to tell the function where to save the file.

int size_x:

The width of the screen shot.

int size_y:

The height of the screen shot.

int start_bar=-1:

The index of the first bar you want to include in your screen shot. 0 means the first visible bar on the chart. The default vaule is -1 which means the end-of-chart screen shot will be taken.

int chart_scale=-1:

The scale (Zoom In and Zoom Out) of the chart screen shot that will be taken. It ranges from 0 to 5. The default value is -1 which means the function will use the current scale of the chart.

int chart_mode=-1

The mode of the chart screen shot that will be taken. it can be one of these values:

CHART_BAR (0)
CHART_CANDLE (1)
CHART_LINE (2)

The default value is -1 which means the function will use the chart mode.

Example:

```
int lasterror=0;
//---- tester has closed one or more trades
if(IsTesting() && ExtTradesCounter<TradesTotal())
{
//---- make screenshot for further checking
if(!ScreenShot("shots\\tester"+ExtShotsCounter+".gif",640,480))
lasterror=GetLastError();
else ExtShotsCounter++;
ExtTradesCounter=TradesTotal();
}
```

WindowFind:

Syntax:

```
int WindowFind(string name)
```

Description:

The *WindowFind* function searches the chart sub windows for the indicator short name passed to it and returns the window index if found and -1 otherwise!

Note: The indicators name can be set using the function IndicatorShortName().

Note: If the indicator search for his name in the init() function using WindowFind() function it always

returns -1.

Parameters:

The function takes only one parameter.

string name:

The name of the indicator short name you want to search for and return its window index.

Example:

```
int win_idx=WindowFind("MACD(12,26,9)");
```

WindowHandle:

Syntax:

```
int WindowHandle(string symbol, int timeframe)
```

Description:

The *WindowHandle* function searches all the opened chart for the symbol (currency pairs) and timeframe passed to it and returns the window handle of the chart window if found and -1 otherwise.

Parameters:

The function takes two parameters:

string symbol:

The symbol name you want to search for and return its window index.

int timeframe:

The timeframe you want to search for and return its window index.

Note: The chart window you are searching must have the symbol name and the timeframe to return the window handle. if the one of the two conditions is found and other not found the return value is -1.

Example:

```
 int win_handle=WindowHandle("EURUSD",PERIOD_H1);
if(win_handle!=0)
Print("Window with EURUSD,H1 detected. Rates array will be copied immediately.");
```

WindowIsVisible:

Syntax:

```
bool WindowIsVisible(int index)
```

Description:

The *WindowIsVisible* function returns true if the chart sub-window index passed to it is visible to the user and false otherwise.

Parameters:

The function takes only one parameter:

int index:

The index of the chart sub-window you want to check its visibility.

Example:

```
 int maywin=WindowFind("MyMACD");
if(maywin>-1 && WindowIsVisible(maywin)==true)
Print("window of MyMACD is visible");
else
Print("window of MyMACD not found or is not visible");
```

WindowOnDropped:

Syntax:

```
int WindowOnDropped()
```

Description:

The *WindowOnDropped* function returns the index of the window you dragged and released the expert advisor, indicator or the script on.

Parameters:

The function doesn't take any parameters.

Example:

```
if(WindowOnDropped()!=0)
{
Print("Indicator 'MyIndicator' must be applied to main chart window!");
return(false);
}
```

WindowsTotal:

Syntax:

```
int WindowsTotal()
```

Description:

The *WindowsTotal* function returns the count of windows (sub-windows and main window) on the chart.

Parameters:

The function doesn't take any parameters.

Example:

```
Print("Windows count = ", WindowsTotal());
```

WindowXOnDropped:

Syntax:

```
int WindowXOnDropped()
```

Description:

The *WindowXOnDropped* function returns the x-axis coordinate in pixels of the point you dragged and released the expert advisor, indicator or the script on.

Parameters:

The function doesn't take any parameters.

Example:

```
Print("Expert dropped point x=",WindowXOnDropped()," y=",WindowYOnDropped());
```

WindowYOnDropped:

Syntax:

```
int WindowYOnDropped()
```

Description:

The *WindowYOnDropped* function returns the Y-axis coordinate in pixels of the point you dragged and released the expert advisor, indicator or the script on.

Parameters:

The function doesn't take any parameters.

Example:

```
Print("Expert dropped point x=",WindowXOnDropped()," y=",WindowYOnDropped());
```

Lesson 17 - Your First Script

It was a long path for you to reach this lesson, Congratulations!

You learnt the basics of MQL4 language then you wrote your first indicator and your first expert advisor.

I hope you enjoyed the journey and interested to continue the road with me.

Today we will create our first script which will simply set the *stoploss* and *takeprofit* values for all the already opened orders on the chart.

It's useful if you are lazy to set manually your *stoploss* and *takeprofit* values from the *Modify or Delete Orders* window for every order.

What are we waiting for? Let's script!

What's a MQL4 script?

The scrip is a program you run once to execute an action and it will be removed immediately after the execution.

Unlike the expert advisors which will be called every time in a new tick arrival, the script will be executed only for once.

And unlike the indicators which have the ability to draw lines on the chart, the script has no access to indicator functions.

In short the script is an expert advisor which you can run only once.

Let's create our first script!

Wizards again!

With the help of the *New Program* wizard we will create the backbone of our scrip.

Bring the *New Program* wizard by choosing *New* from *File* menu or by clicking CTRL+N hot keys (Figure 1).

We are going to create a script, so choose *Scrip program* in the wizard and click *next.*

That will bring the second step wizard (Figure 2).

Fill the Name, Author and Link fields with what you see in the figure 2 (or whatever you want). Then click finish.

Figure 1 – New Program wizard

Figure 2 – Step 2

By clicking *Finish* button the wizard will write some code for you and left the places to write your own code, this is the code we have got from the wizard.

```
//+------------------------------------------------------------------+
//| My_First_Script.mq4 |
//| Copyright Coders Guru |
//| http://www.forex-tsd.com |
//+------------------------------------------------------------------+
#property copyright "Copyright Coders Guru"
#property link "http://www.forex-tsd.com"
//+------------------------------------------------------------------+
//| script program start function |
//+------------------------------------------------------------------+
int start()
{
//----
//----
return(0);
}
//+------------------------------------------------------------------+
```

Note: As you can easily notice from the above code that the wizard hasn't added the *init()* and *deinit()* functions and only added the *start()* function.

That's because it's rare to find a script needs to execute code in the program initialization and de-initialization because the *start()* function itself will be executed for once.

But that's not mean you can't add *init()* and *deinit()* functions in a script. You can add them if you want.

Now, we have to add our code to make our script more useful.

This is the code we have added to the above wizard's generated code (our added code marked by the bold font):

```
//+------------------------------------------------------------------+
//| My_First_Script.mq4 |
//| Copyright Coders Guru |
//| http://www.forex-tsd.com |
//+------------------------------------------------------------------+
#property copyright "Copyright Coders Guru"
#property link "http://www.forex-tsd.com"
#property show_inputs
#include <stdlib.mqh>
extern double TakeProfit=250;
extern double StopLoss=35;
//+------------------------------------------------------------------+
//| script program start function |
//+------------------------------------------------------------------+
int start()
{
//----
int total,cnt,err;
total = OrdersTotal();
for(cnt=0;cnt<total;cnt++)
{
OrderSelect(cnt, SELECT_BY_POS, MODE_TRADES);
if(OrderType()==OP_BUY) // long position is opened
{
OrderModify(OrderTicket(),OrderOpenPrice(),
Bid-Point*StopLoss,Bid+Point*TakeProfit,0,Green);
err=GetLastError();
Print("error(",err,"): ",ErrorDescription(err));
Sleep(1000);
```

```
}
if(OrderType()==OP_SELL) // short position is opened
{
OrderModify(OrderTicket(),OrderOpenPrice(),Ask+Point*StopLoss,
Ask-Point*TakeProfit,0,Red);
err=GetLastError();
Print("error(",err,"): ",ErrorDescription(err));
Sleep(1000);
}
}
//----
return(0);
}
//+------------------------------------------------------------------+
```

Let's crack the code line by line as we used to do.

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

Here the wizard has written the comment block which contains the script name we have been chosen, the copyright and link we have been typed.

Then two directive properties have been added according to the date we have entered, these are the copyright and link properties.

```
#property show_inputs
```

We have added the *show_inputs* directive property to our script.

Without this property the script will not accept inputs from the user therefore will not prompt an input window to the user as the one showed in figure 3.

In our script the user can set the *TakeProfit* and *StopLoss* values from the inputs window

or he can leave the default values we have set in our code.

Note: If you intend to write a script which doesn't need inputs from the user, it's preferred to remove *show_inputs* directive property.

Figure 3 – Script Input window

```
#include <stdlib.mqh>
```

Later in our code we are going to use the function *ErrorDescription* which returns a string description of the passed error number. This function is included in the file *"stdlib.mqh"*, so we have included this file to our program using the *include* directive. That means the content of the *"stdlib.mqh"* is a part of our code now and we can freely use the *ErrorDescription* function.

```
extern double TakeProfit=250;
extern double StopLoss=35;
```

Here we have declared two extern variables which the user can set them from the script inputs window (Figure 2) or he can leave the default values.

We will use these variables in our *start()* function to set the *takeprofit* and *stoploss* values of all the already opened order.

```
int start()
{
//----
int total,cnt,err;
total = OrdersTotal();
....
}
```

We are inside the *start()* function which will be called only one time when you attach the script to you chart.

We have declared three integer variables using a single line declaration method.

Then we assigned the return value of the *OrdersTotal* function to the *total* variable.

As you remember *OrdersTotal* function returns the number of opened and pending orders.

```
for(cnt=0;cnt<total;cnt++)
{
```

....

}

Here we enter a *for* loop starts from 0 and ends to the *total* of already opened and pending orders. We will use the loop variable *cnt* with the *OrderSelect* function to select every opened order by its index. In every cycle of the loop we increase the *cnt* variable by 1.

OrderSelect(cnt, SELECT_BY_POS, MODE_TRADES);

We have selected the order by its index using the function *OrderSelect* and the loop variable *cnt*.

We have to use the *OrderSelect* function before calling *OrderType* in the coming line.

(Please review Appendix 2 – Trading functions).

if(OrderType()==OP_BUY) // long position is opened

{

....

}

The *OrderType* function returns the type of selected order that will be one of:

***OP_BUY, OP_SELL, OP_BUYLIMIT, OP_BUYSTOP, OP_SELLLIMIT* or *OP_SELLSTOP*.**

In the above *if* block we have checked the type of order to find is it a *Buy* order or not.

If it's a *Buy* order, the code inside the block will be executed.

OrderModify(OrderTicket(),OrderOpenPrice(),

Bid-Point*StopLoss,Bid+Point*TakeProfit,0,Green);

It's a *Buy* order type, so we want to modify the values of *stoploss* and *takeprofit* to the values of *StopLoss* and *TakeProfit* variables the user has been set.

We use the *OrderModify* function which modifies the properties of a specific opened order or pending order with the new values you pass to the function.

We used these parameters:

ticket: We've got the current selected order ticket with *OrderTicket* function.

price: We've got the open price of the current selected order with *OrderOpenPrice* function.

stoploss: Here we have set the *stoploss* value of the current selected order to the value of the *subtraction* of the current *bid* price and the *StopLoss* value the user has been set.

takeprofit: Here we have set the *takeprofit* value of the current selected order to the value of the *addition* of the current *bid* price and the *TakeProfit* value the user has been set.

expiration: We haven't set an *expiration* date to our order, so we used 0.

arrow_color: We have set the color of the arrow to *Green*.

```
err=GetLastError();
Print("error(",err,"): ",ErrorDescription(err));
Sleep(1000);
```

We have assigned the returned value of the *GetLastError* function to the *err* variable.

The *GetLastError* returns the number of the last error occurred after an operation (*OrderModify* in our case).

But we want to *print* not only the error number but the description of the error, so we have used the *ErrorDescription* function to get the string description of the error number and we have used the print function to output this message to the expert log.

Then we have used the *Sleep* function to give the terminal and our script the time to take their breath for one second (1000 milliseconds).

Note: *Sleep* function suspends the execution of the program for a specified interval, this interval passed to the function in milliseconds.

```
if(OrderType()==OP_SELL) // short position is opened
{
....
}
```

In the above *if* block we have checked the type of order to find is it a *Sell* order or not.

If it's a *Sell* order, the code inside the block will be executed.

```
OrderModify(OrderTicket(),OrderOpenPrice(),Ask+Point*StopLoss,
Ask-Point*TakeProfit,0,Red);
```

It's a *Sell* order type, so we want to modify the values of *stoploss* and *takeprofit* to the values of *StopLoss* and *TakeProfit* variables the user has been set.

We use the *OrderModify* function again with these parameters:

ticket: We've got the current selected order ticket with *OrderTicket* function.

price: We've got the open price of the current selected order with *OrderOpenPrice*

function.

stoploss: Here we have set the *stoploss* value of the current selected order to the value of the *addition* of the current *ask* price and the *StopLoss* value the user has been set.

takeprofit: Here we have set the *takeprofit* value of the current selected order to the value of the *subtraction* of the current *ask* price and the *TakeProfit* value the user has been set.

expiration: We haven't set an *expiration* date to our order, so we used 0.

arrow_color: We have set the color of the arrow to *Red*.

err=GetLastError();

Print("error(",err,"): ",ErrorDescription(err));

Sleep(1000);

The same as above explained lines.

return(0);

It's the time to terminate our script by using *return* function.

Let's compile our script by pressing F5 hot key, as soon as you compile the script it will appear in the navigator window in the script category (Figure 4)

Note: The wizard has created the file *"My_First_Script.mq4"* and has placed it in the *"experts\scripts"* folder for us.

You have to put all of you scripts in this folder and compile them before using them in the terminal.

Figure 4 – Execute your script

To execute the script simply point it with your mouse and double click it, or click the left mouse button and a menu as showed in figure 4 will appear then choose *Execute On Chart*.

The inputs window (Figure 3) will appear and you can set new values for *StopLoss* and *TakeProfit* or leave the default values.

I hope you enjoyed the lesson and found the first script we have created a useful one.

I welcome very much your questions and suggestions.

Coders' Guru

Lesson 18 - Working with templates

We have created our first indicator, expert advisor and script in the previous lessons.

Every time we decide to create our program whatever the kind of the program (indicator, expert or script) we have to write it from scratch.

This is a useful thing for MQL4 beginners, but it's time consuming for advanced programmers.

That's why MQL4 has the feature of creating *templates based programs*.

Today we will create an indicator based on MACD template shipped with MetaTrader.

So, let's template!

What are the MQL4 templates?

The MQL4 templates are text files contain instructions (commands) for MetaEditor to generate a new program based on these instructions. MetaEditor will read these instructions and use them to generate the appropriated lines of code.

We use the templates to duplicate the frequently used codes and save our coding time.

By using the templates you can create indicators, experts and scripts based on your favorite indicators, experts and scripts and save the time of writing the code from scratch.

Today we are going to duplicate the MACD indicator and we will not add to it anything that's because we are not in the indicator creating lesson but we are in the template lessons.

Inside look:

We are going to give the template file a look before digging into our creation of our first template based indicator.

Note: The template files are stored in the experts\templates path.

And have the ".mqt" file extension.

Here's the content of the "MACD.mqt" template which we will use it today creating our template based indicator:

Note: We have indented some of content and colored them to make it clearer but you can give the original file an inside look.

```
<expert>

type=INDICATOR_ADVISOR

separate_window=1

used_buffers=2
```

```
<param>

type=int

name=FastEMA

value=12

</param>

<param>

type=int

name=SlowEMA

value=26

</param>

<param>

type=int

name=SignalSMA

value=9

</param>

<ind>

color=Silver

type=DRAW_HISTOGRAM

</ind>

<ind>

color=Red

</ind>

</expert>

#header#

#property copyright "#copyright#"

#property link "#link#"

#indicator_properties#

#extern_variables#

#mapping_buffers#

//---- indicator buffers

double ExtSilverBuffer[];
```

```
double ExtRedBuffer[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function |
//+------------------------------------------------------------------+
int init()
{
#buffers_used#;
//---- drawing settings
#indicators_init#
//----
SetIndexDrawBegin(1,SignalSMA);
IndicatorDigits(5);
//---- indicator buffers mapping
SetIndexBuffer(0, ExtSilverBuffer);
SetIndexBuffer(1, ExtRedBuffer);
//---- name for DataWindow and indicator subwindow label
IndicatorShortName("MACD("+FastEMA+","+SlowEMA+","+SignalSMA+")");
//---- initialization done
return(0);
}
//+------------------------------------------------------------------+
//| Moving Averages Convergence/Divergence |
//+------------------------------------------------------------------+
int start()
{
int limit;
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--;
```

```
limit=Bars-counted_bars;
//---- macd counted in the 1-st buffer
for(int i=0; i<limit; i++)
ExtSilverBuffer[i]=iMA(NULL,0,FastEMA,0,MODE_EMA,PRICE_CLOSE,i)-
iMA(NULL,0,SlowEMA,0,MODE_EMA,PRICE_CLOSE,i);
//---- signal line counted in the 2-nd buffer
for(i=0; i<limit; i++)
ExtRedBuffer[i]=iMAOnArray(ExtSilverBuffer,Bars,SignalSMA,0,MODE_SMA,i)
;
//---- done
return(0);
}
//+------------------------------------------------------------------+
```

If you give the above code a look you can notice that the most of the code is a normal MQL4 code with two new kinds of different code.

The first kind of code (Ex: <expert> and <param>) looks like the HTML tags; these are the *template tags*.

And the second kind of code (Ex: #header# and #link#) looks like the MQL4 directives; these are the *template commands*:

MQL4 template tags:

The template file starts with tags very like the Html and XML tags.

There is a start tag and a closing tag. The closing tag uses a slash after the opening bracket. For example <expert> is the start tag and </expert> is the closing tag.

The text between the brackets is called an element. For example:

type=int

name=FastEMA

value=12

There are three tags in MQL4 template:

expert tag:

This is main tag and the other two tags are belonging to it.

In this tag we write the type of program, the indicator chart window, the number of buffer

used, the bottom border for the chart and the top border for the chart of the indicator.

These are the elements used in the expert tag:

type: The type of program. Possible values are EXPERT_ADVISOR,

INDICATOR_ADVISOR, SCRIPT_ADVISOR.

separate_window: The chart window type.

used_buffers: The number of used buffers.

ind_minimum: The fixed bottom border of indicator window.

ind_maximum: The fixed top border of indicator window.

param tag:

We use this tag to create external variables. For every variable we use a param tag.

These are the elements used in the param tag:

type: The data type of the variable.

name: The variable name.

value: The default value of the variable.

ind tag:

We use this tag to set the parameters of every indicator (drawing line) we use in our program.

These are the elements used in the ind tag:

type: The indicator drawing shape style. Possible values are DRAW_LINE,

DRAW_HISTOGRAM, DRAW_SECTION, DRAW_ARROW.

color: The indicator color.

arrow: The character for displaying the DRAW_ARROW indicator. "Wingdings" font is used for displaying the characters.

MQL4 template commands:

They are lines of code starts and ends with # symbol. And these lines of code will be replaced with the corresponded lines with code in the generation process.

Note: MetaEditor reads these lines and replace them in the place they found and generate the MQ4 file.

These are the template commands:

#header#: This line will be replaced with the program header block (comments with file name, author name and the company name).

#copyright#: This line will be replaced with your company name.

#link#: This line will be replaced with your website link.

#indicator_properties#: This line will be replaced with the properties of the indicator.

#extern_variables#: This line will be replaced with external variables used in your program with their types and default values.

#buffers_used#: This line will be replaced with the number of buffer used if any.

#indicators_init#: This line will be replaced with the initialization of the indicators (using the function *SetIndexStyle*).

Creating our MACD template based indicator.

Now, let's create our template based indicator by hitting our *File* menu and choose *New* or clicking CTRL+N. That will bring the New Program wizard (Figure 1).

This time we will choose "Create from template" option and from the drop list we will choose "Indicator – MACD".

Figure 1 – New program wizard

Then we will hit *Next* button which brings the second step wizard (Figure 2).

MetaEditor has filled the fields with *Author* name and the *Link* (reading them from the windows registry) and left the *Name* field blank which we typed in it "MACD_From_Template"

In the *Parameters* list the MetaEditor has red our template file and filled the list with the external variables and its default values.

Figure 2 – Second step wizard

Hitting *Next* button will bring the third step wizard (figure 3) which contains the indicator parameters.

As you guessed MetaEditor has red our template file and generated the values of the indicator properties.

Now click the finish button and you will see the magic of the templates.

Figure 3 – Third step wizard

What have we got?

The wizards have generated a new indicator for us based on MACD template.

Code ready to compile, ready to run:

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

```
//| MACD_From_Template.mq4 |
//| Copyright Coders Guru |
//| http://www.forex-tsd.com |
//+------------------------------------------------------------------+
#property copyright "Copyright Coders Guru"
#property link "http://www.forex-tsd.com"
#property indicator_separate_window
#property indicator_buffers 2
#property indicator_color1 Silver
#property indicator_color2 Red
//---- input parameters
extern int FastEMA=12;
extern int SlowEMA=26;
extern int SignalSMA=9;
//---- buffers
//---- indicator buffers
double ExtSilverBuffer[];
double ExtRedBuffer[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function |
//+------------------------------------------------------------------+
int init()
{
//---- drawing settings
SetIndexStyle(0,DRAW_HISTOGRAM);
SetIndexStyle(1,DRAW_LINE);
//----
SetIndexDrawBegin(1,SignalSMA);
IndicatorDigits(5);
//---- indicator buffers mapping
SetIndexBuffer(0, ExtSilverBuffer);
```

```
SetIndexBuffer(1, ExtRedBuffer);
//---- name for DataWindow and indicator subwindow label
IndicatorShortName("MACD("+FastEMA+","+SlowEMA+","+SignalSMA+")");
//---- initialization done
return(0);
}
//+------------------------------------------------------------------+
//| Moving Averages Convergence/Divergence |
//+------------------------------------------------------------------+
int start()
{
int limit;
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--;
limit=Bars-counted_bars;
//---- macd counted in the 1-st buffer
for(int i=0; i<limit; i++)
ExtSilverBuffer[i]=iMA(NULL,0,FastEMA,0,MODE_EMA,PRICE_CLOSE,i)-
iMA(NULL,0,SlowEMA,0,MODE_EMA,PRICE_CLOSE,i);
//---- signal line counted in the 2-nd buffer
for(i=0; i<limit; i++)
ExtRedBuffer[i]=iMAOnArray(ExtSilverBuffer,Bars,SignalSMA,0,MODE_SMA,i)
;
//---- done
return(0);
}
//+------------------------------------------------------------------+
```

Compile the code and enjoy your new MACD indicator.

I welcome very much your questions and suggestions.

Coders' Guru

Appendix 1 - BARS

MQL4 COURSE

By Coders' guru

(Appendix 1)

Bars

I have got a lot of questions about the mystery of the bars count.

I'm going to describe everything about Bars in this appendix.

What's a BAR?

The bar is the dawning unit on the chart which you can specify by choosing the period of the timeframe.

For example: The 30 minutes timeframe will draw a bar every 30 minutes.

MetaTrader (and other platforms) uses the values of to high, low, open and close prices to draw the bar start and end boundaries. (Figure 1)

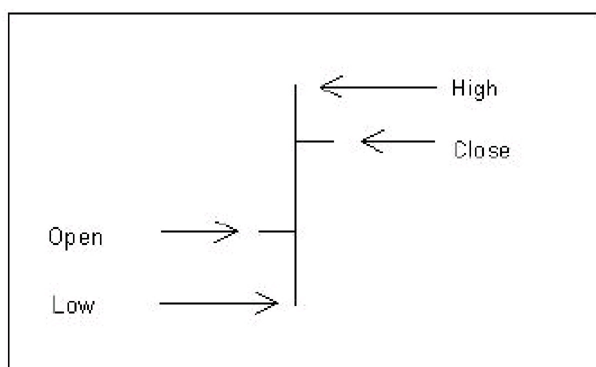


Figure 1 - Bar chart dawning

How the bars indexed in MQL4?

MQL4 indexes the bars from 0 (for the current bar) then 1, 2, 3 etc to the count of the bars.

So, if you want to work with the current bar you use the index 0.

And the index of the previous bar (of the current bar) is 1.

And the index 100 is the bar 100 in the history (100 bars ago).

And the index of last bar is the count of all bars on chart.

MQL4 BARS count functions:

In MQL4 there is a variety of functions working with the count of bars:

int Bars

This function returns the number of the bars of the current chart.

Note that, changing the timeframe will change this count.

int iBars(string symbol, int timeframe)

This function returns the number of bars on the specified currency pairs symbol and timeframe.

Assume you are working on 30M timeframe and you want to get the count of Bars on 1H

time frame, you use this line:

iBars(NULL, PERIOD_H1));

Note: If you used it like this:

iBars(NULL,0));

It returns the same number as Bars function.

int IndicatorCounted()

When you are writing your indicator, you know now how to get the number of bars.

You use this number in your calculation and line dawning.

But it's useful to know if you have counted the bar before or it's the first time you count it.

That's because, if you have counted it before you don't want to count it again and want to

work only with the new bars.

In this case you use the function IndicatorCounted(), which returns the number of bars

have been counted by your indicator.

At the first run of your indicator this count will be zero, that's because your indicator

didn't count any bars yet.

Afterwards, it will equal to the count of Bars – 1. That's because the last bar not counted

yet (Figure 2).

16:55:08	Last Bar Open = 1.18230000
16:55:08	First Bar Open = 1.22200000
16:55:08	Bars = 15831
16:55:08	counted_bars = 15830
16:55:02	Last Bar Open = 1.18230000
16:55:02	First Bar Open = 1.22200000
16:55:02	Bars = 15831
16:55:02	counted_bars = 0

Figure 2 – Program output

Let's write a small program to show you what's going on.

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

//| Bars.mq4 |

//| Codersguru |

//| http://www.forex-tsd.com |

//+------------------------------------------------------------------+

#property copyright "Codersguru"

#property link "http://www.forex-tsd.com"

#property indicator_chart_window

//+------------------------------------------------------------------+

//| Custom indicator initialization function |

//+------------------------------------------------------------------+

int init()

{

//---- indicators

//----

return(1);

}

//+------------------------------------------------------------------+

//| Custor indicator deinitialization function |

//+------------------------------------------------------------------+

int deinit()

{

//----
```

```
//----
return(0);
}
//+------------------------------------------------------------------+
//| Custom indicator iteration function |
//+------------------------------------------------------------------+
int start()
{
//----
Alert( "counted_bars = " , IndicatorCounted()); //The count of bars
have been counted
Alert( "Bars = " , Bars); //The count of the bars on the chart
Alert( "iBars = " , iBars(NULL,0)); //the same as Bars function
Alert( "First Bar Open = " , Open[Bars-1]); //Open price of the
first bar
Alert( "Last Bar Open = " , Open[0]); //Open price of the last bar
(current bar)
//----
return(0);
}
//+------------------------------------------------------------------+
```

Note: This program produces the image you have seen in figure 2

I hope the Bars count is clear now.

I welcome very much the questions and the suggestions._

See you

Coders' Guru

Appendix 2 - Trading Functions

MQL4 COURSE

By Coders' guru

(Appendix 2)

Trading Functions

In this appendix you will find the description of the 25 MQL4 trading functions.

I decided to write this appendix before writing the third part of "Your First Expert Advisor" lesson because you have to know these important functions before cracking the remaining of the code.

OrderSend:

Syntax:

```
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=CLR_NONE)
```

Description:

The *OrderSend* function used to open a sell/buy order or to set a pending order.

It returns the ticket number of the order if succeeded and -1 in failure.

Use *GetLastError* function to get more details about the error.

Note: The ticket number is a unique number returned by *OrderSend* function which you can use later as a reference of the opened or pending order (for example you can use the ticket number with *OrderClose* function to close that specific order).

Note: *GetLastError* function returns a predefined number of the last error occurred after an operation (for example when you call *GetLastError* after *OrderSend* operation you will get the error number occurred while executing *OrderSend*).

Calling *GetLastError* will reset the last error number to 0.

You can find a full list of MQL4 errors numbers in *stderror.mqh* file.

And you can get the error description for a specific error number by using *ErrorDescription* function which defined at *stdlib.mqh* file.

Parameters:

This function takes 11 parameters:

string symbol:
The symbol name of the currency pair you trading (Ex: EURUSD and USDJPY).

Note: Use *Symbol()* function to get currently used symbol and *OrderSymbol* function to get the symbol of current selected order.

int cmd:
An integer number indicates the type of the operation you want to take; it can be one of these values:

Constant	Value	Description
OP_BUY	0	Buying position.
OP_SELL	1	Selling position.
OP_BUYLIMIT	2	Buy limit pending position.
OP_SELLLIMIT	3	Sell limit pending position.
OP_BUYSTOP	4	Buy stop pending position.
OP_SELLSTOP	5	Sell stop pending position.

Note: You can use the integer representation of the value or the constant name.

For example:
***OrderSend(Symbol(),0,…)* is equal to *OrderSend(Symbol(),OP_BUY,…)* .**

But it's recommended to use the constant name to make your code clearer.

double volume:
The number of lots you want to trade.

double price:
The price you want to open the order at.
Use the functions *Bid* and *Ask* to get the current bid or ask price.

int slippage:
The slippage value you assign to the order.

Note: *slippage* is the difference between estimated transaction costs and the amount actually paid. *slippage* is usually attributed to a change in the spread. (*Investopedia.com*).

double stoploss:
The price you want to close the order at in the case of losing.

double takeprofit:
The price you want to close the order at in the case of making profit.

string comment:
The comment string you want to assign to your order (*Figure 1*).
The default value is *NULL* which means there's no comment assigned to the order.

Note: Default value of a parameter means you can leave (don't write) it out, and MQL4 will use a predefined value for this parameter.

For example we can write *OrderSend* function with or without *comment* parameter like this:

OrderSend(Symbol(),OP_BUY,Lots,Ask,3,Ask-25*Point,Ask+25*Point,"My order comment",12345,0,Green);

Or like this:
OrderSend(Symbol(),OP_BUY,Lots,Ask,3,Ask-25*Point,Ask+25*Point,12345,0,Green);

int magic:
The magic number you assign to the order.

Note: Magic number is a number you assign to your order(s) as a reference enables you to distinguish between the different orders. For example the orders you have opened by your expert advisor and the orders have opened manually by the user.

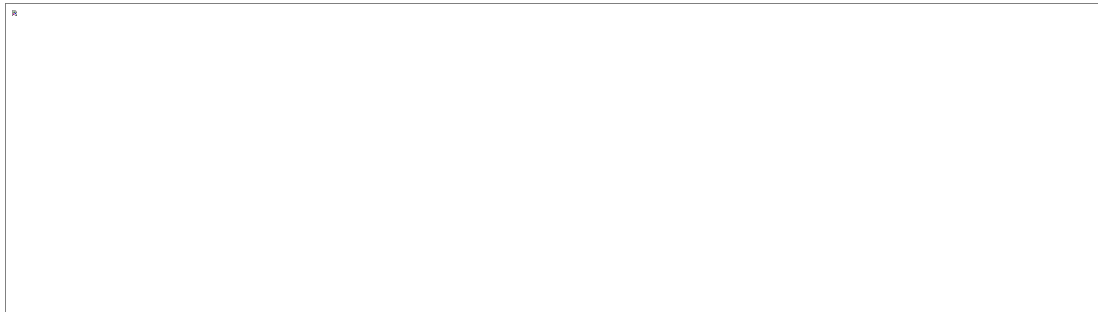


Figure 1 - Comment

datetime expiration:

The time you want your pending order to expire at.

The default time is 0 which means there's no exportation.

Note: The time here is the server time not your local time, to get the current server time use *CurTime* function and to get the local time use *LocalTime* function.

color arrow_color:

The color of opening arrow (*Figure 2*), the default value is *CLR_NONE* which means there's no arrow will be drawn on the chart.

Figure 2 – Arrows color

<u>Example</u>:

```
int ticket;
if(iRSI(NULL,0,14,PRICE_CLOSE,0)<25)
  {
    ticket=OrderSend(Symbol(),OP_BUY,1,Ask,3,Ask-25*Point,Ask+25*Point,"My order #2",16384,0,Green);
    if(ticket<0)
      {
       Print("OrderSend failed with error #",GetLastError());
       return(0);
```

```
    }
}
```

Arrays

Hi folks,

The subject of Arrays in any programming language is a big mystery to any new comer. MQL4 is not exception.
Today we are going to reveal the mystery behind the Array in general and in MQL4 in particular.

What are the Arrays?

In our life we usually group similar objects into units, in the programming we also need to group together the data items of the same type. We use Arrays to do this task.
Arrays are very like the list tables, you group the items in the table and access them by number of the row, but rows in the Arrays called Indexes.
The rows in the Arrays start from 0 not from 1 like anything else. So the index of the first item in an array is 0 and the index of the second item is 1 and the index of the third item is 2 etc.

Note: In some of programming language the Arrays index starts from 1, it called 1 indexed arrays, but in MQL4 and the most of programming languages the Arrays index start from 0 (0 indexed Arrays).

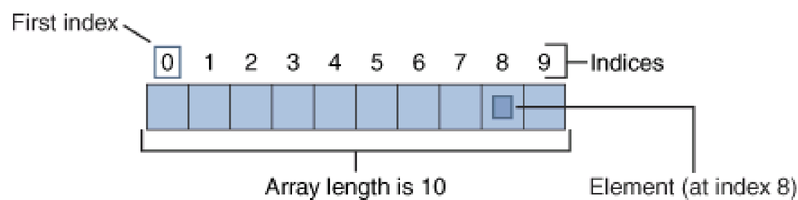


Figure 1 – one dimension array of 10 integers

Array dimensions:

In the most of cases the Array has one dimension like the description above, but in some case the Array has more than one dimension (In MQL4 the maximum dimension is four dimensions).
Multi-dimensional arrays are very like multi-column tables, where the first column is the first dimension of the Array and the second column is the second dimension of the Array etc.

Creating an Array:

The arrays must to be declared (like the variables) before using it in the code. Declaring an array means creating it by giving it a name (identifier), type and size.

In MQL4 we are creating the Arrays like this:

```
int myarray[50];
```

In the code above we declared one-dimensional array.
int keyword indicates that the array is an array of integers.
myarray is the name of the array (identifier).
[] telling the compiler that we are declaring an array not an integer variable.
50 is the size of the array.
We have now a one-dimensional array of 50 integers

```
double myarray[5][40];
```

The line above is the way to declare two-dimensional of seven arrays each of them consisting of 40 doubles.

Initializing the Array:

Initializing an array (or a variable) means setting its value at the same declaration line.
Look at this code which initializes one- dimensional array of 6 integers:

```
int myarray [6] = {1,4,9,16,25,36};
```

The list of array element have to be enclosed between curly braces {} then be assigned to the array.

The number of elements has to be equal to the array size. If you provided number of elements lesser than the array size the last elements of arrays will be filled with zeros. And if you provided number greater than the array size, MQL4 will ignore these elements and set the array to the first elements.

For example the following code will fill the last element of the array with zero because the size of array is 6 elements and we provided it 5 elements:

```
int myarray [6] = {1,4,9,16,25}; //5 elements only for 6 size array.
Alert ("myarray[0] " + myarray[0]);
Alert ("myarray[1] " +myarray[1]);
Alert ("myarray[2] " +myarray[2]);
Alert ("myarray[3] " +myarray[3]);
Alert ("myarray[4] " +myarray[4]);
Alert ("myarray[5] " +myarray[5]);
```

The code above will produce will fill the array with the elements you see in figure 2.

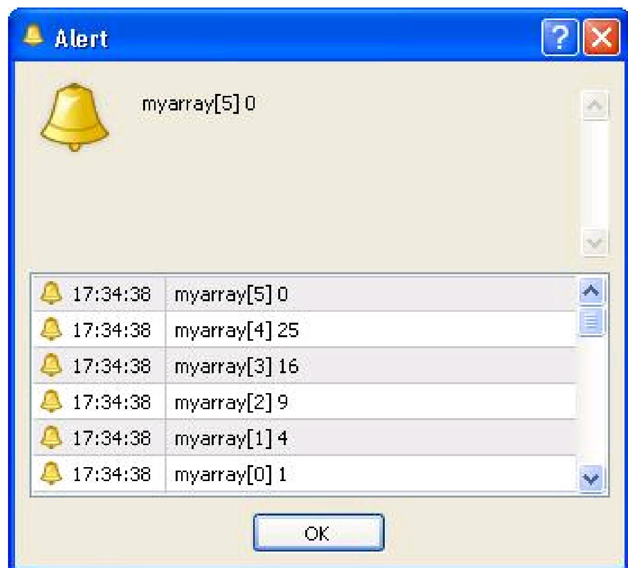


Figure 2 - assigned elements lesser than array size

```
int myarray [6] = {1,4,9,16,25,36,55}; //7 elements for 6 size array.
Alert ("myarray[0] " + myarray[0]);
Alert ("myarray[1] " +myarray[1]);
Alert ("myarray[2] " +myarray[2]);
Alert ("myarray[3] " +myarray[3]);
Alert ("myarray[4] " +myarray[4]);
Alert ("myarray[5] " +myarray[5]);
```

The code above will produce will fill the array with the elements you see in figure 3.

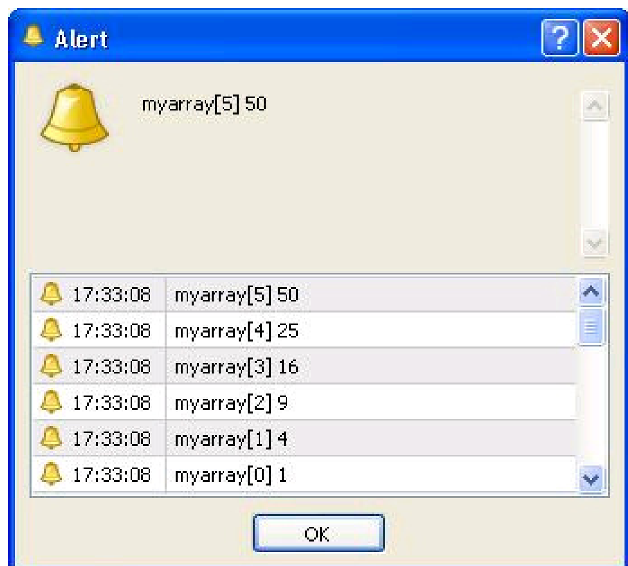


Figure 3 - assigned elements greater than array size.

Assigning elements to an Array:

You've created an array and want to fill it with elements. You can do that in two ways, the first way we have explained above is initializing the array with the elements in the declaration line.

If you didn't initialize the array yet, don't worry. You can assign the elements of the array in two ways:

1- Assigning the elements of the array using the element index.
In this method you assign element by element using the index of the element, The first element is 0 index and the second is 1 index etc.
Look at this code:

```
int myarray [6] ;
myarray[0] = 1;
myarray[1] = 4;
myarray[2] = 9;
myarray[3] = 16;
myarray[4] = 25;
myarray[5] = 36;

Alert ("myarray[0] " + myarray[0]);
Alert ("myarray[1] " + myarray[1]);
 Alert ("myarray[2] " + myarray[2]);
Alert ("myarray[3] " + myarray[3]);
Alert ("myarray[4] " + myarray[4]);
Alert ("myarray[5] " + myarray[5]);
```

In the code above we declared an array without initializing it. Then we set each element with a line of code.

2- Assigning the element of one filled array to an empty array:
In this method you have already filled array and you assign it to another array. The assigned to array must be the same size of the assigned from one.

Look at this code:

```
int mysourcearray [6] = {1,4,9,16,25,36};
int mycopyarray[6] ;
mycopyarray[0] = mysourcearray [0];
mycopyarray[1] = mysourcearray [1];
mycopyarray[2] = mysourcearray [2];
mycopyarray[3] = mysourcearray [3];
mycopyarray[4] = mysourcearray [4];
mycopyarray[5] = mysourcearray [5];

Alert ("mycopyarray[0] " + mycopyarray[0]);
Alert ("mycopyarray[1] " + mycopyarray[1]);
Alert ("mycopyarray[2] " + mycopyarray[2]);
Alert ("mycopyarray[3] " + mycopyarray[3]);
Alert ("mycopyarray[4] " + mycopyarray[4]);
Alert ("mycopyarray[5] " + mycopyarray[5]);
```

Note: You can use MQL4 ArrayCopy function which copies one array to another one. But the ArrayCopy is not the scope of our today article and we will discuss the MQL4 Arrays Function in another article.

Accessing the elements of the Array:

When you create an array and fill it with the elements, you can access the elements of the array by the index number of each element.
The first element in the array has the index 0 and the second has the index 1 etc.
Look at this code:

```
int myarray [6] = {1,4,9,16,25,36 };
Alert ("The first element at myarray is: " + myarray[0]);
Alert ("The second element at myarray is: " + myarray[1]);
Alert ("The third element at myarray is: " + myarray[2]);
Alert ("The fourth element at myarray is: " + myarray[3]);
Alert ("The fifth element at myarray is: " + myarray[4]);
Alert ("The sixth element at myarray is: " + myarray[5]);
```

In the above code myarray[0] is the first element in the array and myarray[1] is the second element etc. The last element in this array is myarray[5]. (Figure 4).

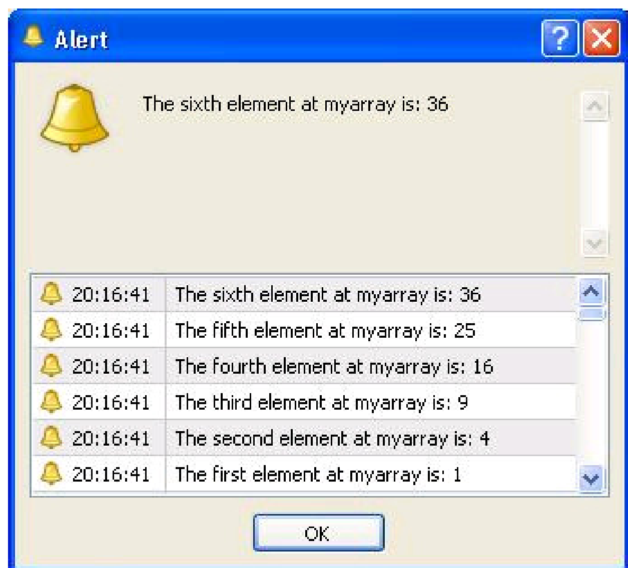


Figure 4 - accessing the elements or the array

You can type code like this to access the third array element:

```
int third_element;
third_element = myarray[2];
Alert(third_element);  //9
```

Hope that the Arrays is understandable subject now. And see you in other article.

Coders Guru

MQL4 Array Functions

Hi folks,

We have talked about the Arrays in a previous article and we covered all the aspects of the Arrays in general and in MQL4 in particular. But MQL4 concerned more about the arrays and for that it built a set of functions handling the arrays.

Today will talk about this set of build-in MQL4 functions one by one.

ArrayBsearch:

Syntax:

int ArrayBsearch(double array[], double value, int count=WHOLE_ARRAY, int start=0, int direction=MODE_ASCEND)

Description:

The ArrayBsearch function searches an array for the occurrence of a value. The function returns the index of the first occurrence of the value if the value is found in the array and returns the nearest one if the value is not found.

The function searches only the first dimension of the array and cannot be used with the arrays of strings or serial numbers.

Note: You have to sort the numeric arrays using *ArraySort()* function before performing binary search.

Parameters:

double array[]
The numeric array you want to search it.

double value
The value you are searching for in the array.

int count
The count of elements in the array you want to search them. By default the function searches the whole of the array (WHOLE_ARRAY or 0).

int start
The index you want to start from your searching. By default, the search starts on the first element (0).

int direction
The searching direction; It can be one of the following values:
MODE_ASCEND: forward direction searching;
MODE_DESCEND: backward direction searching.

Example:

```
datetime daytimes[];
 int shift=10,dayshift;
 // All the Time[] timeseries are sorted in descendant mode
 ArrayCopySeries(daytimes,MODE_TIME,Symbol(),PERIOD_D1);
 if(Time[shift]>=daytimes[0]) dayshift=0;
 else
 {
 dayshift=ArrayBsearch(daytimes,Time[shift],WHOLE_ARRAY,0,MODE_DESCEND);
```

```
 if(Period()<PERIOD_D1) dayshift++;
 }
 Print(TimeToStr(Time[shift])," corresponds to ",dayshift," day bar opened
at", TimeToStr(daytimes[dayshift]));
```

ArrayCopy:

Syntax:

int ArrayCopy(object& dest[], object source[], int start_dest=0, int start_source=0, int count=WHOLE_ARRAY)

Description:

The *ArrayCopy* function copies an array to another array. The source and destination arrays must be of the same type, but arrays with type double[], int[], datetime[], color[], and bool[] can be copied as arrays with same type.

The function returns the number of the elements has been copied.

Parameters:

object& dest[]
The destination array - the array you want to copy to.

object source[]
The source array - the array you want to copy from.

int start_dest
The starting index for the destination array. By default, start index is 0.

int start_source
The starting index for the source array. By default, start index is 0.

int count
The count of elements that should be copied. By default, all the elements of the source array (WHOLE_ARRAY).

Example

```
 double array1[][6];
 double array2[10][6];
 // fill array with some data
 ArrayCopyRates(array1);
 ArrayCopy(array2, array1,0,Bars-9,10);
 // now array2 has first 10 bars in the history
```

ArrayCopyRates:

Syntax:

int ArrayCopyRates(double& dest_array[], string symbol=NULL, int timeframe=0)

Description:

The *ArrayCopyRates* function copies the rate from the chart RateInfo array to a two-dimensional array where the second dimension has these elements.

0 – time;
1 – open;
2 – low;
3 – high;
4 – close;
5 - volume.

Note: The numbers in front of the above rate are the index in the array.

Parameters:

double& dest_array[]
The array you want to copy the chart rate info to.

string symbol
The symbol name of the currency pair you want to get the rate info. Default (NULL) means the current currency used in the chart.

int timeframe
The chart periodicity you want to use. It can be any of the following values:

Constant	Value	Description
PERIOD_M1	1	1 minute.
PERIOD_M5	5	5 minutes.
PERIOD_M15	15	15 minutes.
PERIOD_M30	30	30 minutes.
PERIOD_H1	60	1 hour.
PERIOD_H4		

240

4 hour.

PERIOD_D1

1440

Daily.

PERIOD_W1

10080

Weekly.

PERIOD_MN1

43200

Monthly.

0 (zero)

0

The current Time frame used on the chart.

Use 0 if you want the current timeframe of the chart.

Example:

```
double array1[][6];
ArrayCopyRates(array1,"EURUSD", PERIOD_H1);
 Print("Current bar ",TimeToStr(array1[0][0]),"Open", array1[0][1]);
```

ArrayCopySeries:

Syntax:

int ArrayCopySeries(double& array[], int series_index, string symbol=NULL, int timeframe=0)

Description:

The *ArrayCopyRates* function copies a series array to another array. The function returns the number of the elements has been copied.

Parameters:

double& array[]
The array you want to copy to.

int series_index
The type of the series array you want to copy. It can be any of following values:

Constant

Value

Description

MODE_OPEN

0

Open price.

MODE_LOW

1

Low price.

MODE_HIGH

2

High price.

MODE_CLOSE

3

Close price.

MODE_VOLUME

4

Volume, used in Lowest() and Highest() functions.

MODE_TIME

5

Bar open time, used in ArrayCopySeries() function.

Note: If the series_index is MODE_TIME, the first distension array must be a datetime array.

string symbol
The symbol name of the currency pair you want to get the rate info. Default (NULL) means the current currency used in the chart.

int timeframe
The chart periodicity you want to use. Use 0 if you want the current timeframe of the chart.

Example:

```
datetime daytimes[];
 int shift=10,dayshift;
 // All the Time[] timeseries are sorted in descendant mode
 ArrayCopySeries(daytimes,MODE_TIME,Symbol(),PERIOD_D1);
 if(Time[shift]>=daytimes[0]) dayshift=0;
 else
 {
 dayshift=ArrayBsearch(daytimes,Time[shift],WHOLE_ARRAY,0,MODE_DESCEND);
 if(Period()<PERIOD_D1) dayshift++;
 }
 Print(TimeToStr(Time[shift])," corresponds to ",dayshift," day bar opened at ",
```

TimeToStr(daytimes[dayshift]));

ArrayDimension:

Syntax:

int ArrayDimension(int array[])

Description:

The *ArrayDimension* function returns the dimensions count of the given array.

Parameters:

int array[]
The array you want to retrieve its dimensions count.

Example:

```
int num_array[10][5];
 int dim_size;
 dim_size=ArrayDimension(num_array);
 // dim_size is 2
```

ArrayGetAsSeries:

Syntax:

bool ArrayGetAsSeries(object array[])

Description:

The *ArrayGetAsSeries* function checks the elements of given array and retruns true if the array is a series array (array elements indexed from last to first) and false otherwise.

Parameters:

int array[]
The array you want to check its elements.

Example:

```
if(ArrayGetAsSeries(array1)==true)
 Print("array1 is indexed as a series array");
 else
 Print("array1 is indexed normally (from left to right)");
```

ArrayInitialize:

Syntax:

int ArrayInitialize(double& array[], double value)

Description:

The *ArrayInitialize* function sets all the elements of the given numeric array to the same value. The function returns the count of the elements in the created array.

Parameters:

double& array[]
The numeric array you want to initialize.

double value
The value you want to set all the elements of the array to.

Example:

```
//---- setting all elements of array to 2.1
 double myarray[10];
 ArrayInitialize(myarray,2.1);
```

ArrayIsSeries:

Syntax:

bool ArrayIsSeries(object array[])

Description:

The *ArrayIsSeries* function checks the given array if it's a series array (time, open, close, high, low, or volume) or not. The function returns true if the array is a series array and false otherwise.

Parameters:

object array[]
The array you want to check.

Example:

```
if(ArrayIsSeries(array1)==false)
 ArrayInitialize(array1,0);
 else
 {
 Print("Series array cannot be initialized!");
 return(-1);
 }
```

ArrayMaximum:

Syntax:

int ArrayMaximum(double array[], int count=WHOLE_ARRAY, int start=0)

Description:

The *ArrayMaximum* function searches the elements of the given array for the maximum value. The function returns the index of the maximum value

Parameters:

double array[]
The array you want to search.

int count

The count of elements in the array you want to search them. By default the function searches the whole of the array (WHOLE_ARRAY or 0).

int start
The index you want to start from your searching. By default, the search starts on the first element (0).

Example:

```
double num_array[15]={4,1,6,3,9,4,1,6,3,9,4,1,6,3,9};
 int maxValueIdx=ArrayMaximum(num_array);
 Print("Max value = ", num_array[maxValueIdx]);
```

ArrayMinimum:

Syntax:

int ArrayMinimum (double array[], int count=WHOLE_ARRAY, int start=0)

Description:

The *ArrayMinimum* function searches the elements of the given array for the minimum value. The function returns the index of the minimum value

Parameters:

double array[]
The array you want to search.

int count
The count of elements in the array you want to search them. By default the function searches the whole of the array (WHOLE_ARRAY or 0).

int start
The index you want to start from your searching. By default, the search starts on the first element (0).

Example:

```
double num_array[15]={4,1,6,3,9,4,1,6,3,9,4,1,6,3,9};
 int maxValueIdx=ArrayMaximum(num_array);
 Print("Max value = ", num_array[maxValueIdx]);
```

ArrayRange:

Syntax:

int ArrayRange(object array[], int range_index)

Description:

The *ArrayRange* function returns the count of the elements of the given dimension of the given array.

Note: Because the arrays in MQL4 are zero-based array, the size of the array is the largest index + 1.

Parameters:

object array[]
The array you want to check.

int range_index
The dimension you want to check.

Example:

```
int dim_size;
 double num_array[10,10,10];
 dim_size=ArrayRange(num_array, 1);
```

ArrayResize:

Syntax:

int ArrayResize(object& array[], int new_size)

Description:

The *ArrayResize* function sets a new size for the first dimension of the given array. The function returns the count of the elements of the resized array if it successfully resized the array and 0 otherwise.

Parameters:

Object& array[]
The array you want to resize.

int new_size
The new size.

Example:

```
double array1[][4];
 int element_count=ArrayResize(array, 20);
 // element count is 80 elements
```

ArraySetAsSeries:

Syntax:

bool ArraySetAsSeries(double& array[], bool set)

Description:

The *ArraySetAsSeries* function reverses the index order of the given array to a series array. The function returns true in success and false otherwise.

Parameters:

double & array[]
The array you want to reverse its elements.

bool set
The series flag to set (true) or drop (false).

Example:

```
double macd_buffer[300];
 double signal_buffer[300];
```

```
 int i,limit=ArraySize(macd_buffer);
 ArraySetAsSeries(macd_buffer,true);
 for(i=0; i<limit;
i++) macd_buffer[i]=iMA(NULL,0,12,0,MODE_EMA,PRICE_CLOSE,i)-iMA(NULL,0,26,0,MODE_EMA,P
 for(i=0; i<limit; i++)
 signal_buffer[i]=iMAOnArray(macd_buffer,limit,9,0,MODE_SMA,i);
```

ArraySize:

Syntax:

int ArraySize(object array[])

Description:

The *ArraySize* function returns the size of the given array.

Parameters:

object array[]
The array (any type) you want to get its size.

Example:

```
int count=ArraySize(array1);
 for(int i=0; i<count; i++)
 {
 // do some calculations.
 }
```

ArraySort:

Syntax:

int ArraySort(double& array[], int count=WHOLE_ARRAY, int start=0, int sort_dir=MODE_ASCEND)

Description:

The *ArraySort* function sorts the first dimension of the given numeric array.

Note: Series arrays can't be sorted by this function.

Parameters:

double& array[]
The numeric array you want to sort.

int count
The count of elements in the array you want to sort them. By default the function sorts the whole of the array (WHOLE_ARRAY or 0).

int start
The index you want to start from your sorting from. By default the sort starts from the first element (0).

int sort_dir
The sorting direction; It can be one of the following values:
MODE_ASCEND: sort ascending;

MODE_DESCEND: sort descending.

Example:

```
double num_array[5]={4,1,6,3,9};
 // now array contains values 4,1,6,3,9
 ArraySort(num_array);
 // now array is sorted 1,3,4,6,9
 ArraySort(num_array,MODE_DESCEND);
 // now array is sorted 9,6,4,3,1
```

I hope you enjoyed the article!
Coders Guru

iCustom function mystery

Hi folks,

I have received a lot of questions in the forums about the iCustom function and today I'll try to reveal the mystery behind the very important MQL4 function; iCustom.

There are two kinds of indicators you can use in your code (Expert advisors, Custom indicators and scripts):

Built-in indicators:

The MQL4 has a number of built-in indicators which you can use them in your code directly as a function for example:

```
double iMA( string symbol, int timeframe, int period, int ma_shift, int ma_method, int applied_price, int shift)
```

The above code calculates the moving average indicator and returns its value.

```
double iATR( string symbol, int timeframe, int period, int shift)
```

The above code calculates the average true range indicator and returns its value.

Any other indicator:

To use any other indicator in your code you have to duplicate the code of this indicator in your code (it the hell way) or you can use the iCustom function:

```
double iCustom( string symbol, int timeframe, string name, ... , int mode, int shift)
```

What's iCustom anyway?

iCustom is a MQL4 function enables you to use external indicators in your expert advisor or custom

indicator code without re-writing the code from scratch.
If you didn't have the code of the external indicator you want to use in your code iCustom is the only way to use the indicator in your code because iCustom works with the already compiled indicator (.exe4 format).

How to use iCustom?

Let's assume that you have built your custom indicator in MQL4 and want to use it in your expert advisor code, and you are not interested in duplicating the indicator code in your expert advisor.

Your indicator draws only one line on the chart. This line is the EMA of the period the user enters (external variable). The code of you custom indicator will be something like that:

```
//+------------------------------------------------------------------+
//|  Demo_Indicator.mq4 |
//|  Codersguru |
//|  http://www.metatrader.info |
//+------------------------------------------------------------------+
#property copyright "Codersguru"
#property link "http://www.metatrader.info" #property indicator_chart_window
#property indicator_buffers 1
#property indicator_color1 Red

//---- inputs
extern int UsePeriod = 13; //---- buffers
double ExtMapBuffer1[];//+------------------------------------------------------------------+
//| Custom indicator initialization function |
//+------------------------------------------------------------------+
int init()
{
//---- indicators
SetIndexStyle(0,DRAW_LINE);
SetIndexBuffer(0,ExtMapBuffer1);
//----
return(0);
}

//+------------------------------------------------------------------+
//| Custor indicator deinitialization function |
//+------------------------------------------------------------------+
int deinit()
{
//----
 //----
return(0);
}

//+------------------------------------------------------------------+
//| Custom indicator iteration function |
//+------------------------------------------------------------------+
int start()
{
int counted_bars=IndicatorCounted();
if (counted_bars<0) return(-1);
//---- last counted bar will be recounted
```

```
  if (counted_bars>0) counted_bars--;
int pos=Bars-counted_bars; //---- main calculation loop
while(pos>=0)
  {
  ExtMapBuffer1[pos]= iMA(NULL,0,UsePeriod,0,MODE_EMA,PRICE_CLOSE,pos);
pos--;
  }
//----
 return(0); }
//+------------------------------------------------------------------+
```

You've compiled the code above and loaded the indicator in MetaTrader terminal and have got this result (Figure 1).

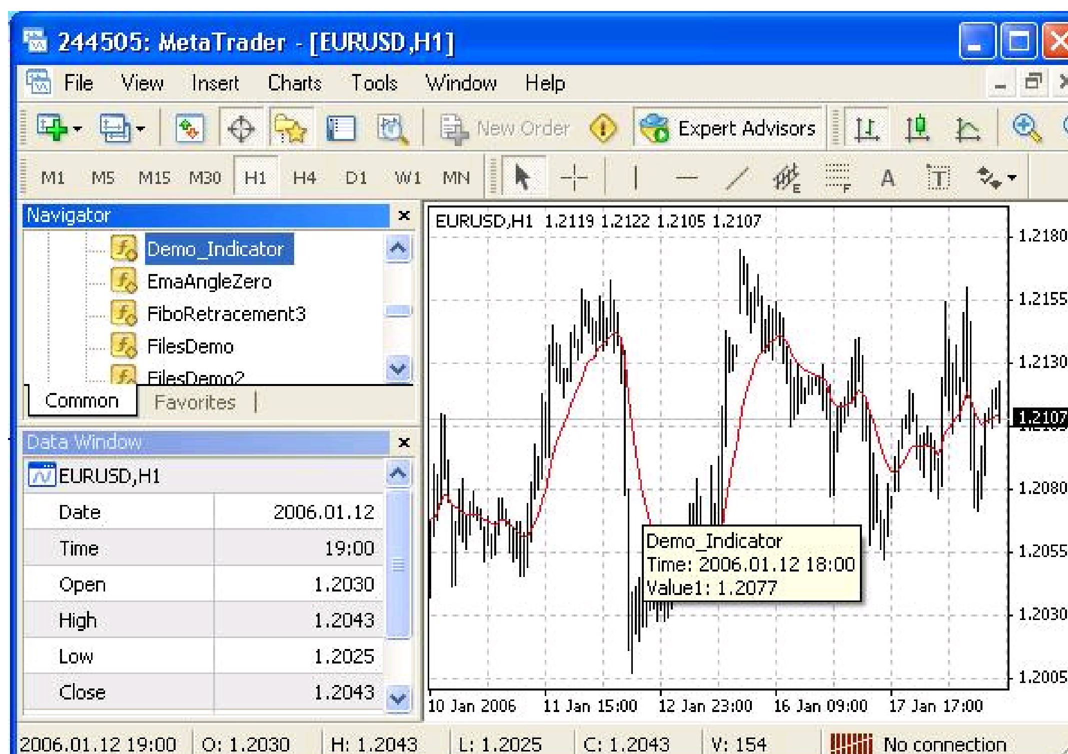


Figure 1 - Demo_Indicator

Just keep in your mind that.
1- Your indicator uses only one external variable; UsePeriod.
2- And draws only one line; ExtMapBuffer1

Now you want to write your expert advisor which uses the indicator above.
Let's assume that you want to use two of your Demo_Indicator.
The first one uses 13 UsePeriod as a parameter and the second one uses 20 UsePeriod as parameter.
At the crossing of the two indicators you will take a trade action (sell or buy).

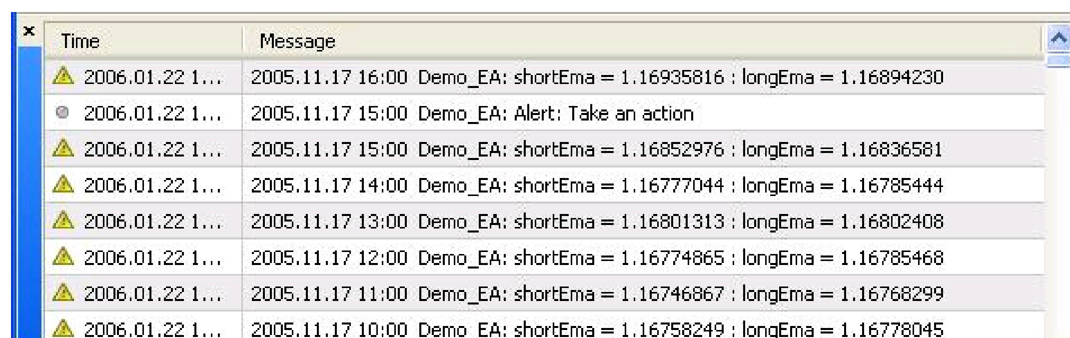
Your code of the expert advisor will be something like that:

```
//+------------------------------------------------------------------+
//| Demo_EA.mq4 |
//| Codersguru |
```

```
//| http://www.metatrader.info |
//+------------------------------------------------------------------+

#property copyright "Codersguru"
#property link "http://www.metatrader.info"

//+------------------------------------------------------------------+
//| 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 total;
double shortEma, longEma;

shortEma = iCustom(NULL,0,"Demo_Indicator",13,0,0);
```

```
longEma = iCustom(NULL,0,"Demo_Indicator",20,0,0);
Print("shortEma = " + shortEma + " : longEma = " + longEma);

int isCrossed = 0;
isCrossed = Crossed (shortEma,longEma);
total = OrdersTotal();
if(total < 1)
{
if(isCrossed == 1)
{
Alert("Take an action");
return(0);
}

if(isCrossed == 2)
{
Alert("Take an action");
return(0);
}
return(0);
}
return(0);
}
//+------------------------------------------------------------------+
```

Compile the above code and load the expert advisor in terminal then go to the Strategy Tester (press F6). Test your expert advisor and notice the journal tab (Figure 2).
The lines:
Print("shortEma = " + shortEma + " : longEma = " + longEma);
And
Alert("Take an action");
Printed in the journal as you see in figure 2.



Time	Message
2006.01.22 1...	2005.11.17 16:00 Demo_EA: shortEma = 1.16935816 : longEma = 1.16894230
2006.01.22 1...	2005.11.17 15:00 Demo_EA: Alert: Take an action
2006.01.22 1...	2005.11.17 15:00 Demo_EA: shortEma = 1.16852976 : longEma = 1.16836581
2006.01.22 1...	2005.11.17 14:00 Demo_EA: shortEma = 1.16777044 : longEma = 1.16785444
2006.01.22 1...	2005.11.17 13:00 Demo_EA: shortEma = 1.16801313 : longEma = 1.16802408
2006.01.22 1...	2005.11.17 12:00 Demo_EA: shortEma = 1.16774865 : longEma = 1.16785468
2006.01.22 1...	2005.11.17 11:00 Demo_EA: shortEma = 1.16746867 : longEma = 1.16768299
2006.01.22 1...	2005.11.17 10:00 Demo_EA: shortEma = 1.16758249 : longEma = 1.16778045

Figure 2 - Demo_EA
We have used the iCustom function in the above code to get the calculated value of our indicator, let's give iCustom an inside look.

iCustom syntax:

double iCustom(string symbol, int timeframe, string name, ... , int mode, int shift)

iCustom function takes these parameters:

string symbol:

**The symbol name of the currency pair you trading (Ex: EURUSD and USDJPY).
Use NULL for the current symbol.**

ote: Use *Symbol()* function to get currently used symbol and *OrderSymbol* function to get the symbol of current selected order.

**int timeframe:
The chart periodicity you want to use. It can be any of the following values:**

Constant	Value	Description
PERIOD_M1	1	1 minute.
PERIOD_M5	5	5 minutes.
PERIOD_M15	15	15 minutes.
PERIOD_M30	30	30 minutes.
PERIOD_H1	60	1 hour.
PERIOD_H4	240	4 hour.
PERIOD_D1	1440	Daily.
PERIOD_W1		

10080

Weekly.

PERIOD_MN1

43200

Monthly.

0 (zero)

0

The current Time frame used on the chart.

string name:
The exact name of the indicator you want to use. The indicator must be placed in the indicators folder and be compiled (you are loading the ex4 not the mql4 file, so you have to compile the indicator before loading it).

…:
The parameters set for the indicator you are calling. (…) means it the parameter can be any type and it can supply any count of parameters.
In our Demo Indicator we have only one parameter (External variable) UsePeriod – an integer type. We set its value here to 13 in the first iCustom call and to 20 in the second call.

int mode:
The index of the line you want to get its value. You know that any indicator can use up to 8 lines (buffers). The index of these lines starts at 0 and up to 7.
In our Demo Indicator we have only one line (buffer) ExtMapBuffer1. the index of this line is 0 (because it's the first line and the only one too).

int shift:
The number of the shifts (backwards) from the current bar you want to calculate.
If you use 0 for the shift value it means you want to work with the current bar without shifting.

iCustom function returns double data type. It returns the calculation value of the passed parameters for the loaded indicator.

How did we use the iCustom function in our code?

```
double shortEma, longEma;

shortEma = iCustom(NULL,0,"Demo_Indicator",13,0,0);
longEma = iCustom(NULL,0,"Demo_Indicator",20,0,0);
```

In the first line we have declared two double variable (shortEma and longEma) which hold the returning value of iCustom.
Then we called iCustom in the second line with these parameters:

parmeter 1 :the symbol - NULL for current symbol.
parmeter 2 : time frame - 0 for current time frame.
parmeter 3 : indicator name - here it's " Demo_Indicator".
parmeter 4 : this is a setting for Demo_Indicator - UsePeriod = 13.
parmeter 5 : the line number (range from 0 to 7) - usually used 0.

parmeter 6 : the working bar - 0 for the current bar.

And in the third line we used iCustom again to get the second indicator value. These are the parameters we used:

parmeter 1 :the symbol - NULL for current symbol.
parmeter 2 : time frame - 0 for current time frame.
parmeter 3 : indicator name - here it's " Demo_Indicator".
parmeter 4 : this is a setting for Demo_Indicator - UsePeriod = 20.
parmeter 5 : the line number (range from 0 to 7) - usually used 0.
parmeter 6 : the working bar - 0 for the current bar.

Now we have the value of the indicator for the current bar, we can use it very like if we have the indicator in our program.

I hope it's clearer now and you can iCustom function without problems.

External functions

Hi folks,

Today we are going to talk about a very important development issue; The external functions in MQL4.

We all know the normal functions in MQL4, if you don't know what are the functions (or you forgot) I'll repeat some my speech about the functions I told you in my MQL4 Functions lesson.

What are the MQL4 functions anyway?

The function is very like the sausage machine, you input the meat and spices in the machine from one side and it outputs the sausage.

The meat and the spices are the function parameters and the sausage is the function output (return value). The machine itself is the function body.

There's only one difference between the functions and the sausage machine that some of functions will return nothing (nothing in MQL4 called void).

Let's take an example:

```
double //type of sausage -return value type
my_func(double a , double b , double c)  //function name and parameters list (meat & spices)
{
 return (a*b+c); //sausage outputs - returned value
}
```

As you see above the function starts with the type of the returned value "double" followed by the function name which followed by parentheses. Inside the parentheses you put the meat and the spices, sorry, you put the parameters of the function.

Here we have put three parameters double a, double b and double c.
Then the function body starts and ends with braces. In our example the function body will produce the operation (a*b+c).
The return keyword is responsible about returning the final result.

You call the above function like this:

double result = my_func(1.5 , 2 , 5.9);

As you see in the above line of code, you first declared a double variable to hold the returning value of our function. Then you assigned to that variable the function calling. You called the function by writing the function name then passed to it between the brackets the proper parameters (three double values in our case).

When MQL4 see the function name, it will take the provided parameters and goes to the function body to execute the code inside that body and return with the value and place it at the same point of the function calling.

This was a summery explaining of the MQL4 function. What about the external functions in MQL4?

External functions:

External functions are functions reside at external files not the source file you are writing right now.
You can use these function by importing them from the external files to your code.
Those external files can be one of two kinds of files:

1- MQL4 executable files (ex4):

You can use the functions reside at an already compiled MQL4 program called library programs. The file must be located at the MetaTrader 4\experts\libraries path and must be compiled (.ex4) before importing the functions you want from them. We will know everything about using the functions reside at these types of MQL4 programs in this article. The creating of this type of program is the subject of another article.

2- Windows Dlls:

DLLs (Dynamic Link Libraries) are programs written in an advanced programming language (like Visual basic or C++) for Microsoft Windows operating system. They are very like the normal programs like the Microsoft Word or FireFox browser but they mostly have no user interface and the other important difference that they can loaded from more than one application.
They are mostly contains shared function which can be used with all the windows application.
Using DLLs from MQL4 give the language a very strong advantage and enables it to make a lot of things another applications can do.
For example you can use the windows winsock DLL to send and receive data through the internet. The options are very wide and even dangerous.
We will know everything about using the functions reside at windows DLLs in this article. The creating of these DLLs is out scope of our MQL4 development field.

Using external functions from a library file:

I've create a library file in the purpose of this article (The steps creating of this file is out or article's scope). Download the file if you want and save it the libraries path then compile it:

icon mylib.zip (309 B)

```
//+------------------------------------------------------------------+
//| mylib.mq4 |
//| Copyright Coders Guru |
//|  http://www.metatrader.info |
//+------------------------------------------------------------------+
#property copyright "Copyright Coders Guru"
#property link "http://www.metatrader.info"
#property library
//+------------------------------------------------------------------+
//| My functions  |
```

```
//+------------------------------------------------------------------+
 int MyExtFunction(int value,int value2)
   {
    return(value+value2);
   }
//+------------------------------------------------------------------+
```

In the above code we have created a normal function MyExtFunction which we will import it from another MQL4 program and use it. The MyExtFunction function takes two integers parameters and returns the value of those two numbers addition.

Let's create a simple program which imports and uses the external function MyExtFunction. Download this file if you want and place it in the scripts folder then compile it:

icon Exteranl functions 1.zip (406 B)

```
//+------------------------------------------------------------------+
//| Exteranl functions 1 .mq4 |
//|  Copyright Coders Guru |
//| http://www.metatrader.info |
//+------------------------------------------------------------------+
#property copyright "Copyright Coders Guru"
#property link "http://www.metatrader.info"

#import "mylib.ex4"
  int MyExtFunction(int value,int value2);
#import

//+------------------------------------------------------------------+
//| script program start function |
//+------------------------------------------------------------------+
int start()
  {
//----
  int result = MyExtFunction(10,20);
  Alert ( result);
//----
   return(0);
 }
//+------------------------------------------------------------------+
```

In the code above to use an external function resides in a MQL4 library file we have taken two steps:
1- Importing the external function from the library.
2- Using (calling) the function like any MQL4 function.

Importing the external function:

To be able to use the external function from your code you have to import it to your code. Importing the function takes three lines of code as you see above:

#import "mylib.ex4"

This is the first import line, we use the #import keyword followed by the library name, the line doesn't end with semi-colon.

int MyExtFunction(int value,int value2);

Then in the second line you declare the function very like the normal functions by writing the type of returned value followed by the function name then the parameters of the function and their types. Without this declaration your program will not know anything about the external function. You can import as many function as you want, in this case you write every function in a separate line. The declaration line must ends with semi-colon.

#import

To tell the compiler of MQL4 that you have finished your importing you have to write another line contains the #import keyword, but in this case without an external file.

Note: You use only one #import line to end all the opened import lines, for example:

```
#import "mylib.ex4"
  int MyExtFunction(int value,int value2);
#import "mylib2.ex4"
  double MyExtFunction2(double value3);
#import
```

Calling the external function:

If you have written right import block lines like the above description, you can call the external function very like calling any normal function in MQL4 program. In our program we called the MyExtFunction like this:

```
int result = MyExtFunction(10,20);
Alert ( result);
```

We passed the numbers 10 and 20 to the function and assigned the returned value to the integer variable result. The Alert function will pop up the result variable like figure 1.

Using external functions from a DLL file:

Importing an external function from a DLL file is the same as importing them from a library file. The only difference is the close import line, in the case of importing an external function from a DLL you have not to write the #import line.

Let's take an example:

Download the file if you want and save it the scripts path then compile it:

icon Exteranl functions 2.zip (439 B)

```
//+------------------------------------------------------------------+
//| Exteranl functions 2.mq4 |
//| Copyright Coders Guru |
//| http://www.metatrader.info |
//+------------------------------------------------------------------+



#property copyright "Copyright Coders Guru"
#property link "http://www.metatrader.info"

#import "user32.dll"
  int MessageBoxA(int hWnd,string lpText,string lpCaption,int uType);

//+------------------------------------------------------------------+
//| script program start function |
//+------------------------------------------------------------------+



int start()
  {
//----
  int result = MessageBoxA(NULL,"Helo world!","MQL4 Messagebox",0);
//----
   return(0);
 }
//+------------------------------------------------------------------+
```

As you can notice in the program above the import lines are the same as the using external function from a library except the closing #import line.
In the above code we used a function of user32.dll the very important windows DLL, and called it in the code very like any normal function.

The MessageBoxA function will pop up a message box like the one in figure 2.

Figure 2 - Message box

MQL2 to MQL4 conversion and vice versa!

Hi folks,

In these series of articles I'll try to remove the stumbling blocks from the road of the programmers who want to convert MQL2 programs to MQL4 and vice versa.

A lot of people find that the MQL2 and the MQL4 are different worlds, because the syntax of MQL2 is based on EasyLanguage while the syntax of MQL4 is very like the C syntax (Besides the new added features to the MQL4 language).

But I think that, although it's hard to convert between the two languages but it's not an impossible task. That's because the both of the two languages talking about the same thing; about creating a program for MetaTrader, so, they are not two separated worlds but they are two faces of the same coin.

We can't set a series of lessons to study MQL2 like which we assigned to MQL4 but you will know a lot of MQL2 principals while you are skimming these conversion articles.

We will take somewhat a practical approach studding the difference between the two languages, that's by studding a real example of a program wrote in the both language and observing the conversion operation has been occurred.

Let's converting!

Our example program:

We are going to work in these articles with an indicator called "3D Oscilator".
Download the MQL2 source code: icon 3D Oscilator_MQL2.zip (974 B)
Download the MQL4 source code: icon 3D Oscilator_MQL4.zip (1009 B)

Note: The 3D Oscilator has been created in MQL2 by *Luis Damiani* and converted to MQL4 by *Ricardo. Ramdass.*

Loading the 3D Oscilator(s) in the MT4 and MT3 terminals:

If you are a lucky person like me and have the both of MT4 and MT3 installed in your machine, you can compile the 3D Oscilator and load it in the Terminals. (If you don't have MT3 yet, you can download it from the MetaQuotes download section: http://www.metaquotes.net/downloads/).

To compile the MQL4 version:

1- Place the "3D Oscilator.mq4" file in the Indicators path.

2- Double click it to open it in MetaEditor.

3- Press F5 to compile the program (produced file extension is .ex4).

4- Open MetaTrader and from the Navigator window navigate to the Custom Indicators section and double click the 3D Oscilator.

5- When you get the Properties window (Figure 1) click OK.

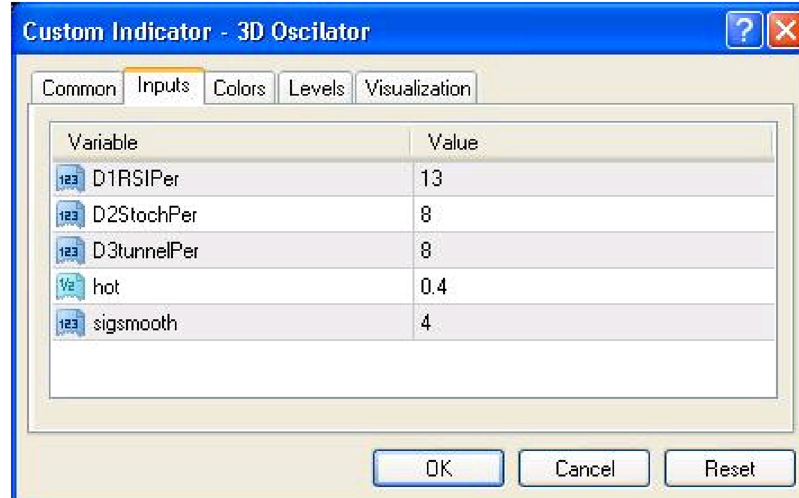


Figure 1 - Properties window in MetaTrader 4

To compile the MQL2 version:

The same steps as MQL4, but you will get a Properties window like this:

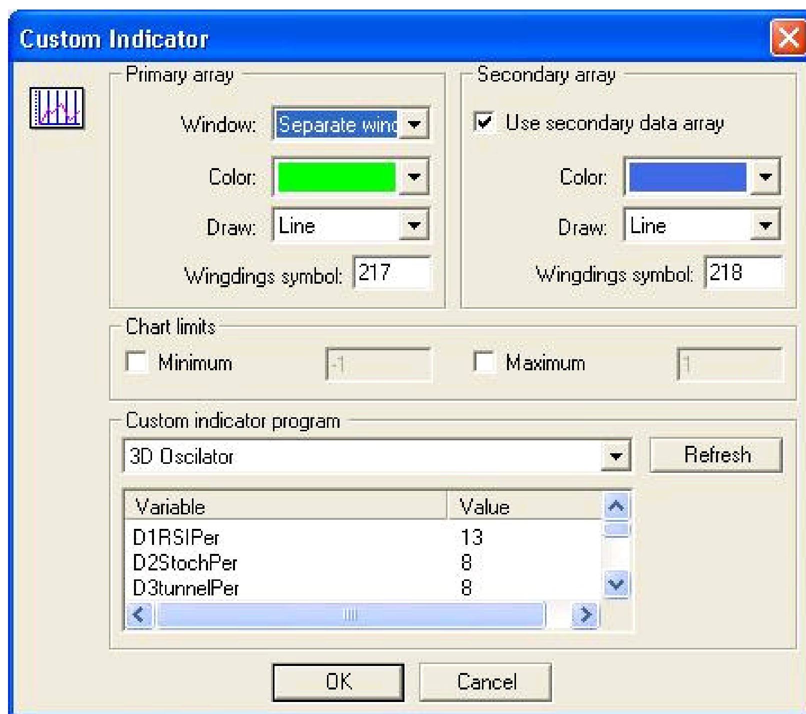


Figure 2 - Properties window in MetaTrader 3

Figures 3 and 4 shows how the indicator looks like in the both platforms:

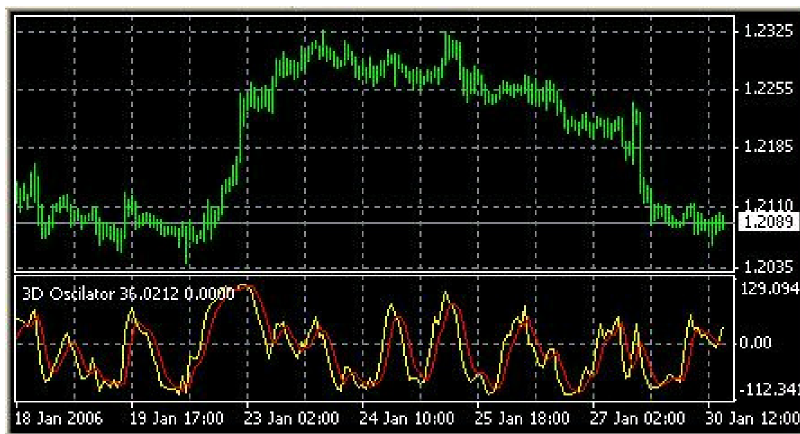


Figure 3 - How the indicator appears MetaTrader 4

Figure 4 - How the indicator appears MetaTrader 3

Cracking the MQL2 program:

We are going to crack the MQL2 program section by section and line by line to convert it to the MQL4 program.

```
 /*[[
 Name := 3D Oscilator
 Author := Luis Damiani
 Separate Window := Yes
 First Color := lime
 First Draw Type := Line
 First Symbol := 217
 Use Second Data :=yes
 Second Color := RoyalBlue
 Second Draw Type := Line
 Second Symbol := 218
]]*/
```

The section above is the description section where you put the name of the program, the author name, window type of the indicator, the indicator line properties and symbols etc.

The section when converted to MQL4 it will be dispersed in the program and not in a gathered section like MQL2.

In this section you can find the following commands:

Name: The name of the program.
Author: The author name of the program.
Separate Window: Plot the indicator in the main chart window (No) or in a separate window.
First Color: The color of the first line (Second, Third etc).
First Draw Type: The type of the drawing line; it maybe: Line, Histogram or Symbol (Second, Third etc).
First Symbol: The symbol number from Wingdings table of the drawing line (Second, Third etc).
Use Second Data: Yes means that the indicator calculation results in 2 charts, and not the only one.
Minimum Chart Limits: If the indicator is shown in a separate window this command sets the minimum value drawn on the chart.
Maximum Chart Limits: the maximum value drawn on the chart.

Let's see how we converted these lines to MQL4:

Name := 3D Oscilator

The Name of the program usually wrote in the comment section in MQL4 along with the Author name and the website Link.

```
//+------------------------------------------------------------------+
//| 3D Oscilator.mq4 |
//| |
//| |
//+------------------------------------------------------------------+
```

Author := Luis Damiani

The Author name wrote in the comment section and in the copyright property directive:

```
#property copyright "Author - Ricardo. Ramdass - Conversion only"
```

Separate Window := Yes

The equivalent line in MQL4 is the indicator_separate_window property:

```
property indicator_separate_window
```

First Color := lime

The color of the line in MQL4 set by the indicator_colorN property, where the N is is the number of the line, for example:

```
#property indicator_color1 //First line color
#property indicator_color2 //Second line color
#property indicator_color8 //Eighth line color
```

In MQL4 we have to add extra line before the indicator_color property, this is the indicator_buffers property which sets the number of the buffers used in the program, the buffers used for the calculations of the indicators and for drawing lines.

In our program we will use two lines and will not use any buffers for calculation. So we have to set 2 buffers:

```
#property indicator_buffers 2
```

And to set the color of the first line we used this line of code:

```
#property indicator_color1 Yellow
```

First Draw Type := Line

To set the drawing type of the line in MQL4 you use SetIndexStyle() function and you have to use it inside the Init() function.

In our program we used code like this:

```
int init() {
 ....
  SetIndexStyle(0,DRAW_LINE);
    ....
 }
```

SetIndexStyle:

void SetIndexStyle(int index, int type, int style=EMPTY, int width=EMPTY, color clr=CLR_NONE)

This function will set the style of the drawn line.

The index parameter of this function ranges from 1 to 7 (that's because the array indexing start with 0 and we have limited 8 lines). And it indicates which line we want to set its style.

The type parameter is the shape type of the line and can be one of the following shape type's constants:

DRAW_LINE (draw a line)
DRAW_SECTION (draw section)
DRAW_HISTOGRAM (draw histogram)
DRAW_ARROW (draw arrow
DRAW_NONE (no draw)

The style parameter is the pen style of drawing the line and can be one of the following styles' constants:

STYLE_SOLID (use solid pen)
STYLE_DASH (use dash pen)
STYLE_DOT (use dot pen)
STYLE_DASHDOT (use dash and dot pen)
STYLE_DASHDOTDOT (use dash and double dots)

Or it can be EMPTY (default) which means it will be no changes in the line style.

The width parameter is the width of line and ranges from 1 to 5. Or it can be EMPTY (default) which means the width will not change.

The clr parameter is the color of the line. It can be any valid color type variable. The default value is CLR_NONE which means empty state of colors.

MQL4 Special functions init(), deinit() and start():

Functions are blocks of code which like a machine takes inputs and returns outputs.In MQL4 there are three special functions which you can't name your functions the same names and have special jobs in MQL4

init():
Every program will run this function before any of the other functions, you have to put here you initialization values of you variables.

start():
Here's the most of the work, every time a new quotation have received your program will call this function.

deinit():
This is the last function the program will call before it shutdown, you can put here any removals you want.

Alerts