Review

Expert Advisors are used to automate the trading process and relieve the trader of the routine functions of continuous market monitoring. Many professional traders employ multiple trading systems enabling them to operate in diverse markets and under a variety of environments. Usually they write and test their trading strategies in the well-known analytical packages, such as MetaStock or TradeStation.

With MetaTrader expert advisor there is a way, whereby you can link the signals generated by the trading systems with your real account, and link them in such a way as to be able to track and manage your open positions, placed orders and stops at any given moment.

What is an Expert Advisor?

It is a mechanical trading system (MTS) written in specialized language MetaQuotes Language II (MQL II) and linked to a particular chart. An Expert Advisor is able not only to notify the trader of the trading opportunities, but also to automatically make deals in the trading account, sending them directly to the trading server. Like most IT systems, Expert Advisors supports the testing of strategies with historical data, with the trade entry/exit points being represented on the charts. Furthermore, the executable code of the Expert Advisor is stored separately from its source text - this arrangement guarantees the concealment (if necessary) of the logic used by the trader from prying eyes.

Writing your own Expert Advisor is very easy: to do it, you don't need to be a professional programmer, you only need to learn to use a very simple language - the MQL II. Even is the user is not able to write the rules for his Expert Advisor on his own, he can always engage an acquaintance of his with decent programming skills, who most likely won't need more than an hour to master the rules and write the programme.

There is a great variety of trading strategies developed by numerous traders for MetaStock and TradeStation. Most of these are easily translated into the MQL II language, which allows the user to incorporate the previously accumulated experience.

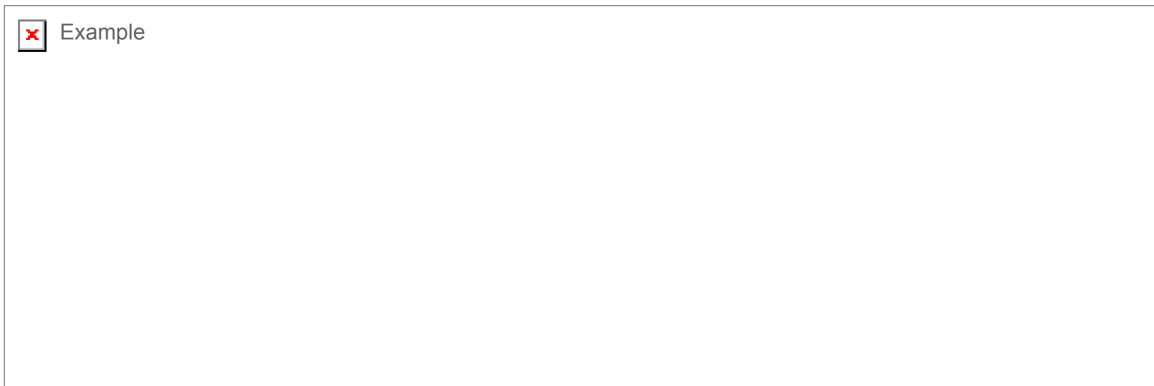
MetaTrader stores Expert Advisors as ***.MQL**(source text) and ***.EXP** (executable code) files in **/Experts** subdirectory of the root directory of the programme. A trader may have an unlimited number of Expert Advisors which can be easily managed through the Navigator window. Procedure of creating the custom Expert Advisors and linking them to the trade terminal is described in detail in the MetaTrader User Guide.

Next >>

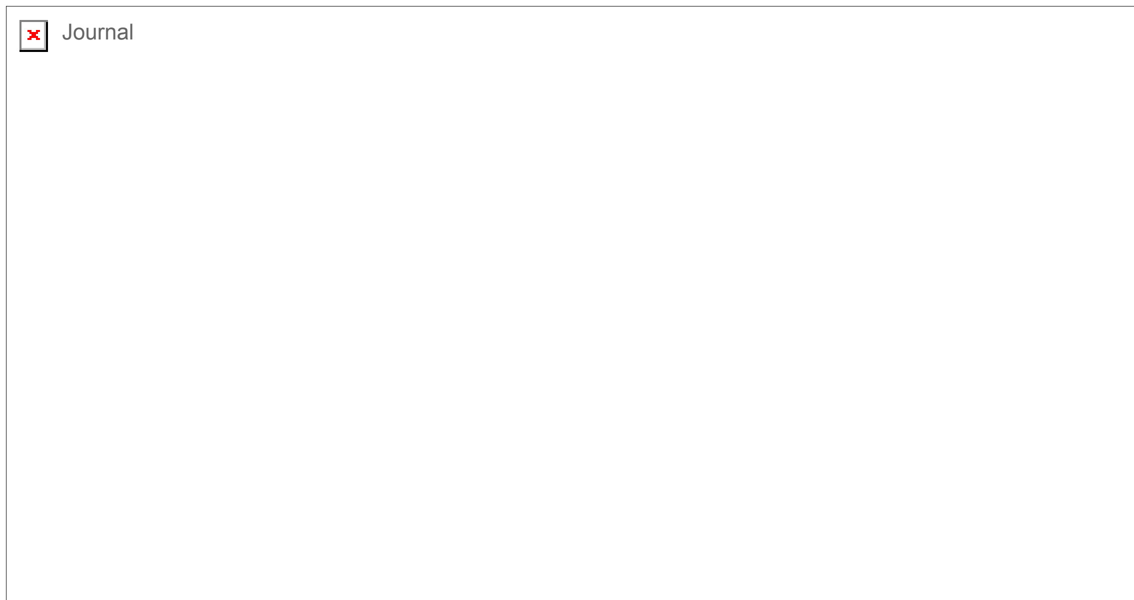
MetaQuotes Language II Syntax

MetaQuotes Language II is used to write custom Expert Advisors to automate the trading process management and implement the trader's own strategies. MetaQuotes Language II is easy to learn, use and set up. MQL II language includes a large number of variables used to control the current and past quotes, principal arithmetic and logical operations and features main built-in indicators and commands used for opening and controlling of the positions. In its syntax the language is similar to the EasyLanguage developed by TradeStation Technologies, Inc., but it also features some specific characteristics.

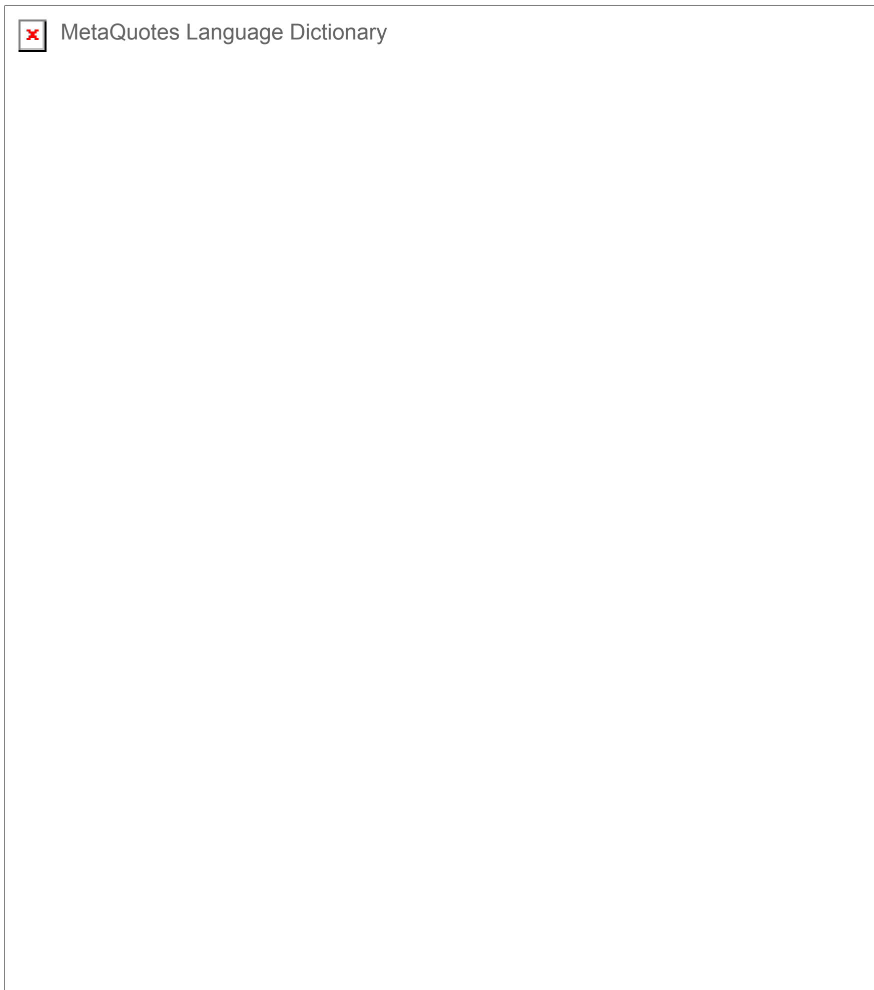
The programme code is written using the **MetaEditor** advisor text editor, which is capable of highlighting different structures of MQL II language in different colors, thus helping the user through the text of the expert system. Comments start with the // symbol (double slash). Comments can also be marked with the 'slash-asterisk' - 'asterisk-slash' pair (/*[comments]*/, as in "C" programming language). The built-in editor highlights comments in gray color.



To set up and control his operational strategy, a trader maintains a log file holding information about the generated signals, variables output and the results of executed trades. Expert Advisor logs are kept in the **/logs/YYYYMMDD.log** file in the MetaTrader folder. The current log may be accessed directly from lower "Terminal" window (Journal tab).



To access the directory system of the MQL II language, the **MetaQuotes Language Dictionary** window is called, either by pressing the Dictionary button or from the Tools menu. The short manual contains the functions split by categories, operations, reserved words etc., and enables the user to get the description of each element used by the language.



1. Main language structures

As any other language, MQL II has a set of main components constituting its basic structure. These components have to be organized and arranged in a particular way, so as to represent proper statements and expressions.

Main object of the language is the data, which may be of 3 types: numerical, logical or string. All the numerical values take the double format. Logical data may take True or False values. A string is a range of characters marked with apostrophes. A character string is also called a text string. Data may be contained in the variables of appropriate types or be represented directly in the source text of the programme.

A MetaQuotes Language statement is a complete instruction. Statements may contain reserved words, operators, data, variables, expressions or punctuation symbols and always end with a semicolon. Reserved words are the predefined words with specific or special meaning. Operators are the symbols designating specific operations on the data, variables and (or) expressions. Variables are used to hold numerical, string or logical data. Expression is a combination of reserved words, variables, data and operators having as a result a value of one of the 3 types used in the language: numerical, logical or character string. Punctuation symbols are used to represent expressions, define parameters, divide words or rearrange the sequence of computations.

2. Punctuation symbols

Character	Name	Description

;	semicolon	Ends an instruction in MetaQuotes Language II
( )	parentheses	Group the values in an expression to change the order of calculation.Mark the parameters in functions and initializing expressions in the descriptions of variables.Mark the initializing values for variables and arrays in the variable description section.
,	comma	Divides the parameters when the functions are called.Divides the variables in the variable description section.Divides the numbers in the description of the array lengths.Divides the indices for accessing the array elements.
:	colon	Is used in the variable description section to start the variable list.
" "	quotation marks	Mark a text (character) string.
[ ]	square brackets	Mark the numbers to specify the array length.Mark numbers (indices) for accessing a particular element of an array.Mark the number of the period for accessing the historical data.
{ }	curly brackets	Serve as operator brackets. May be used instead of begin…end. Isolate a range of instructions into a block.
/* */	comment brackets	Mark the comments.
//	double slash	Specify the start of a single-string comment.

3. Operators

Operators are divided into 5 groups: assignment operators, string operators, mathematical operators, relative operators and logical operators.

3.1. Assignment operator

The assignment operator '=' (the "equal" sign) is used to assign specific values (numerical, string or logical, depending on the variable type) to variables. The assigned value may be a result of an expression. Example:

```
Variable: Counter(0);
…
Counter = Counter + 1;
```

As a result, the Counter variable will take the value 1. Values may also be assigned to array elements.

3.2. String operator

To manipulate text strings, only one operator can be used: '+' (the "plus" sign). It is used to join two strings. Example:

```
Variable: String(" ");
…
String = "some_" + "text";
```

As a result, the String variable will contain the "some_text" text string. Joining strings with numerical and logical values is also permitted. In the latter case the numerical and/or logical values, before joining, will be transformed into the string type. Example:

```
String = "string" + 1;
```

As a result, the String variable will contain the "string1" text string. Operands may be not only values, but also the variables of corresponding types holding such values, as well as expressions which, after they are executed, produce such values.

3.3. Mathematical operators

Four main mathematical operations: addition - '+' ("plus" sign); subtraction - '-' ("minus" sign); multiplication - '*' (asterisk); division - '/' (slash) - are used in the mathematical expressions to calculate the numerical values. Examples of mathematical expressions:

```
( Ask + Bid ) / 2 , High[1] + 20 * Point
```

3.4. Relative operators

Relative operators are used to compare two values of the same type. The first value is compared with the second, resulting in logical values "true" or "false", "less than" - '<' (left brocket); greater than - '>' (right brocket); "equal to" - '=' ("equal" sign); "not equal to" - '<>'; "less than or equal to" - '<='; "greater than or equal to" - '>='.

The logical values obtained as a result of a relative expression are used in the control structures of MetaQuotes Language II. Example:

if FreeMargin < 1000 then exit;

The text strings are compared in lexicographic order, i.e. "aaa" string is considered less than string "zzz". When logical values are compared, one should keep in mind that numerical value of the logical value "True" is 1, while the numerical value of the logical value "False" is 0.

3.5. Logical operators

Logical operators enable the user to combine logical values. The logical OR - '|' (vertical line, or broken bar); logical AND - '&' (ampersand); logical NOT - '!' (exclamation mark). Logical operators have corresponding reserved words OR, AND, NOT. Example:

```
If FreeMargin > 100 and FreeMargin < 1000 then print( "Free margin is ", FreeMargin );
```

Note that, while OR and AND operations are dyadic, that is, they operate with two values, the NOT operation is one-place, that is, it applies to a single value only. Example:

```
Variable: Condition1( True );
…
Condition1 = FreeMargin >= 1000;
If not Condition1 then exit;
```

Below are the tables of results of logical operations.

Value1	Value2	Value1 OR Value2
True	True	True
True	False	True
False	True	True

False	False	False
Value1	Value2	Value1 AND Value2
True	True	True
True	False	False
False	True	False
False	False	False
Value1	Value2	NOT Value1
True		False
False		True

4. Reserved words

MetaQuotes Language II uses several groups of reserved words.

1. Logical operations

    AND, NOT, OR.

2. MQL II commands

    Array, Begin, Break, Continue, Define, Downto, Else, End, Exit, For, If, Input, Then, To, Variable, While.

Reserved words defining the structure of commands of the language are also called keywords.

3. Built-in functions

    Abs, AccName, AccountName, Alert, ArcCos, ArcSin, ArcTan, Ceil, CloseOrder, Comment, Cos, CurTime, Day, DayOfWeek, DelArrow, DeleteOrder, DelGlobalVariable, DelObject, Exp, FileClose, FileDelete, FileOpen, FileReadNumber, FileReadString, FileSeek, FileSize, FileTell, FileWrite, Floor, GetGlobalVariable, GetIndexValue, GetIndexValue2, GetTickCount, Highest, Hour, iAC, iAD, iADX, iADXEx, iAlligator, iAO, iATR, iBands, iBandsEx, iBearsPower, iBullsPower, iBWMFI, iCCI, iCCIEx, iCustom, iDeMarker, iEnvelopes, iForce, iFractals, iGator, iIchimoku, iMA, iMAEx, iMACD, iMACDEx, iMFI, iMom, iMomEx, iOBV, iOsMA, iRSI, iRSIEx, iRVI, iSAR, iSTO, iWPR, IsDemo, IsFileEnded, IsFileLineEnded, IsGlobalVariable, IsIndirect, IsTesting, IsTradeAllowed, LastTradeTime, Log, Lowest, MarketInfo, Max, Min, Minute, Mod, ModifyOrder, Month, MoveObject, Normalize, NumberToStr, Ord, OrderValue, Period, Pow, Print, PrintTrade, Return, Rand, Round, Seconds, ServerAddress, SetArrow, SetDebugMode, SetGlobalVariable, SetIndexValue, SetIndexValue2, SetLoopCount, SetObjectText, SetOrder, SetText, Sin, Sqrt, Srand, StrToTime, Symbol, Tan, TimeDay, TimeDayOfWeek, TimeHour, TimeMinute, TimeMonth, TimeSeconds, TimeToStr, TimeToStrEx, TimeYear, UserFunction, Year.

4. Predefined user variables (user-defined variables)

    Lots, StopLoss, TakeProfit, TrailingStop.

5. Predefined variables of the trading terminal

    AccNum, AccountNumber, Ask, Balance, Bars, Bid, Close, Credit, Equity, FreeMargin, High, Low, Margin, Open, Point, PriceAsk, PriceBid, PriceHigh, PriceLow, PriceTime, Time, TotalProfit, TotalTrades, Volume.

6. Predefined parameters of built-in functions (macros)

MODE_CLOSE, MODE_DETAILS, MODE_EMA, MODE_FILE, MODE_FIRST, MODE_GATORJAW, MODE_GATORLIPS, MODE_GATORTEETH, MODE_HIGH, MODE_KIJUNSEN, MODE_LWMA, MODE_LOW, MODE_LOWER, MODE_MAIN, MODE_MINUSDI, MODE_OPEN, MODE_PLUSDI, MODE_SECOND, MODE_SENKOUSPAN, MODE_SENKOUSPANA, MODE_SENKOUSPANB, MODE_SIGNAL, MODE_SMA, MODE_SMMA, MODE_STOPLOSS, MODE_TAKEPROFIT, MODE_TENKANSEN, MODE_TIME, MODE_UPPER, MODE_VALUES, MODE_VOLUME, OBJ_FIBO, OBJ_HLINE, OBJ_SYMBOL, OBJ_TEXT, OBJ_TRENDLINE, OBJ_VLINE, OP_BUY, OP_BUYLIMIT, OP_BUYSTOP, OP_SELL, OP_SELLLIMIT, OP_SELLSTOP, PRICE_CLOSE, PRICE_HIGH, PRICE_LOW, PRICE_MEDIAN, PRICE_OPEN, PRICE_TYPICAL, PRICE_WEIGHTED, SEEK_CUR, SEEK_END, SEEK_SET, STYLE_SOLID, STYLE_DASH, STYLE_DOT, STYLE_DASHDOT, STYLE_DASHDOTDOT, SYMBOL_ARROWDOWN, SYMBOL_ARROWUP, SYMBOL_CHECKSIGN, SYMBOL_STOPSIGN, SYMBOL_THUMBSDOWN, SYMBOL_THUMBSUP, TIME_DATE, TIME_MINUTES, TIME_SECONDS, VAL_CLOSEPRICE, VAL_CLOSETIME, VAL_COMMENT, VAL_COMMISSION, VAL_LOTS, VAL_OPENPRICE, VAL_OPENTIME, VAL_PROFIT, VAL_STOPLOSS, VAL_SWAP, VAL_SYMBOL, VAL_TAKEPROFIT, VAL_TICKET, VAL_TYPE.

Actually, the above reserved words are macros, i.e. the syntax analyzer replaces them with numerical values. Macros were introduced to simplify the process of writing the function calls for the user: more convenient and mnemonically significant words can be used instead of the numerical values of parameters of some functions. The same applies to the names of colors.

7. Colors

AliceBlue, AntiqueWhite, Aqua, Aquamarine, Azure, Beige, Bisque, Black, BlanchedAlmond, Blue, BlueViolet, Brown, BurlyWood, CadetBlue, Chartreuse, Chocolate, Coral, CornflowerBlue, Cornsilk, Crimson, Cyan, DarkBlue, DarkCyan, DarkGoldenrod, DarkGray, DarkGreen, DarkKhaki, DarkMagenta, DarkOliveGreen, DarkOrange, DarkOrchid, DarkRed, DarkSalmon, DarkSeaGreen, DarkSlateBlue, DarkSlateGray, DarkTurquoise, DarkViolet, DeepPink, DeepSkyBlue, DimGray, DodgerBlue, FireBrick, FloralWhite, ForestGreen, Fuchsia, Gainsboro, GhostWhite, Gold, Goldenrod, Gray, Green, GreenYellow, Honeydew, HotPink, IndianRed, Indigo, Ivory, Khaki, Lavender, LavenderBlush, LawnGreen, LemonChiffon, LightBlue, LightCoral, LightCyan, LightGoldenrod, LightGreen, LightGrey, LightPink, LightSalmon, LightSeaGreen, LightSkyBlue, LightSlateGray, LightSteelBlue, LightYellow, Lime, LimeGreen, Linen, Magenta, Maroon, MediumAquamarine, MediumBlue, MediumOrchid, MediumPurple, MediumSeaGreen, MediumSlateBlue, MediumSpringGreen, MediumTurquoise, MediumVioletRed, MidnightBlue, MintCream, MistyRose, Moccasin, NavajoWhite, Navy, OldLace, Olive, OliveDrab, Orange, OrangeRed, Orchid, PaleGoldenrod, PaleGreen, PaleTurquoise, PaleVioletRed, PapayaWhip, PeachPuff, Peru, Pink, Plum, PowderBlue, Purple, Red, RosyBrown, RoyalBlue, SaddleBrown, Salmon, SandyBrown, SeaGreen, Seashell, Sienna, Silver, SkyBlue, SlateBlue, SlateGray, Snow, SpringGreen, SteelBlue, Tan, Teal, Thistle, Tomato, Turquoise, Violet, Wheat, White, WhiteSmoke, Yellow, YellowGreen.

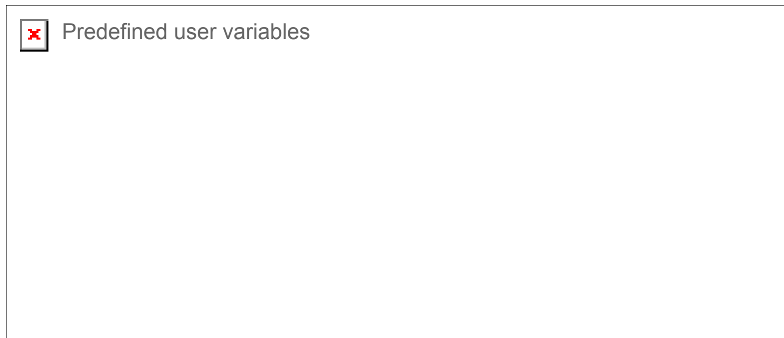
All the reserved words are case-insensitive, i.e. they may be written in lower as well as in upper case.

4.1. Predefined user variables

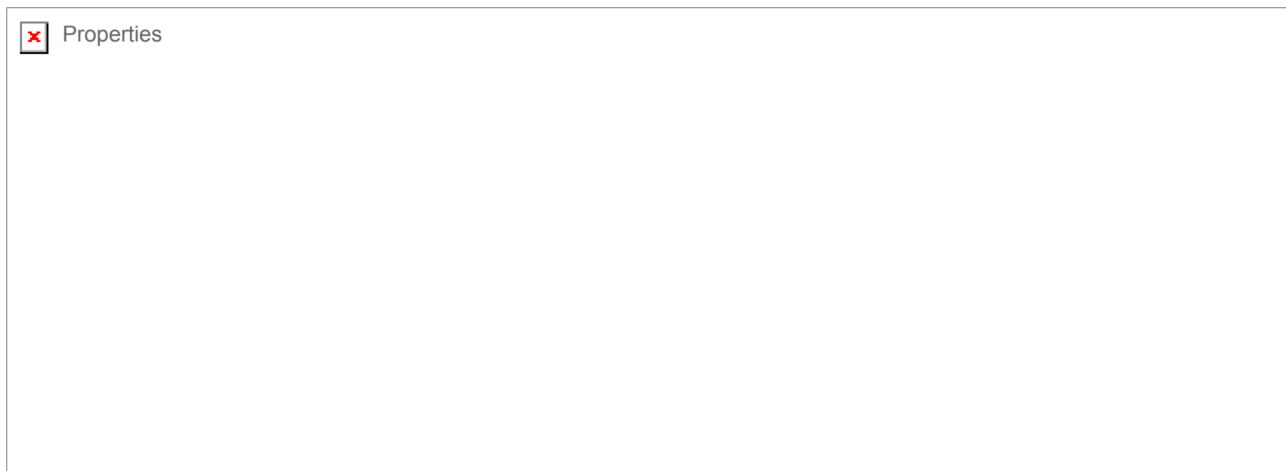
It is often necessary to change certain parameters in an already written Expert Advisor which affect its operation. In order to avoid manual editing of the Advisor code and its critical variables each time such changes are made, an approach similar to that used in the MetaQuotes system was applied: four parameters were taken out to the Properties table (Menu Files of the MetaEditor - Properties... - Processing tab) of the Expert Advisor:

Lots	- number of lots
StopLoss	- Stop Loss level in points
TakeProfit	- Take Profit level in points

TrailingStop - Trailing Stop level in points


Predefined user variables

Initial values of these variables may be entered through the Properties table of the Expert Advisor or changed in a special settings dialog box (Settings tab) called by pressing F7 or via the menu (Charts - Expert Advisors - Properties). These variables can't be modified from the programme.


Properties

4.2. Command structures of MetaQuotes Language II

4.2.1. Variables declaration and description

Storage of the temporary data in the process of calculation requires the use of variables. The variables are described in the very beginning of the programme using reserved words Variable, Array, Define and Input. It is allowed to use these words in plural, i.e. Variables, Arrays, Defines and Inputs. One of these words always starts the variable declaration statement. The difference between these words is the following: "variable" describes simple variables, "array" describes arrays, "define" describes additional user-defined variables which similar to the predefined user variables, may be modified via the Settings dialog box i.e. from the outside. However unlike the predefined variables the user variables may be changed in the process of calculation within the programme. "Input" describes variables used as input parameters for user functions or custom indicator programs.

Variable declaration syntax:

```
Variable : Name( InitialValue );
```

where "Name" is the name of the variable;
"InitialValue" is the initial value of the variable.
The initial value defines the type of the variable - numerical, string or logical.

Example of declaration of a numerical variable:

```
Variable : Counter( 0 );
```

Example of declaration of a string variable:

```
Variable : String1( "some string" );
```

Example of declaration of a logical variable:

```
Variable : MyCondition( false );
```

Array declaration syntax:

```
Array : ArrayName[Array length]( InitialValue );
```

where the array length is denoted by one or several (up to four) numbers - number[, number [, number [, number]]].

In MetaQuotes Language II arrays may be of one-, two-, three- or four-element lenghts. In general terms, an array is a set of variables positioned in a row one after the other which may be accessed by using the same name - the array name - and giving the sequential number (numbers) of the element in an array. Arrays are convenient for holding sequences of data of the same type. A good example of arrays is the historical data of the trading terminal, such as Close, Open, High, Low, Volume. These data are accessed as a single-element array. For example, Close[5] is the value of Close five periods back. Example of a two-element array may be a simple table where the first measurement is the rows and the second measurement - the columns. Example:

```
Array : MyTable[ 10, 5 ]( 0 ); // a 10 rows by 5 columns table
...
print( MyTable[ 2, 4 ] ); // print the fourth element in the second row
```

Arrays may contain values of any type - numerical, string or logical, however, it should be of the type, the initial value of which was specified when the array was declared.

Additional user-defined variable declaration syntax:

```
Define : Name( InitialNumber );
```

where Name is the name of the variable;
InitialNumber is the Initial numerical value of the variable.

It should be noted that the additional user-defined variables may be of numerical type only. As already mentioned above, the user-defined variables may be modified during the calculation process of the programme. Such modifications will only apply in the current session of the Expert Advisor, until the particular Expert Advisor is deleted from the chart or the operation of the customer terminal ends. When the new session of the Expert Advisor starts, the values of the user-defined variables will be initialized anew.

Note that initialization of the all variables (Inputs, Variables, Defines, Arrays) is passed one time only at the first launch of an expert adviser. If you want so that some variable has a specified value at each launch then assign such value explicitly. For example:

```
Variable : cnt( 0 );
cnt = 0; // variable assignment at each launch
```

4.2.2. EXIT statement

The Exit statement breaks the operation of the Expert Advisor. It is the so-called pre-scheduled termination of the programme.

4.2.3. IF-THEN conditional statement

The If-Then conditional statement makes it possible to control the sequence of execution of the Expert Advisor instructions. This statement may be written in different ways. Syntax:

```
if Condition then Statement;
```

or

```
if Condition then begin
Statement;
Statement;
…
end;
```

where Condition is a logical expression taking the values True or False;

"Statement" is any instruction of the MetaQuotes Language II. Operator brackets Begin - End may be replaced with curly brackets { }. The conditional statement can be used to branch the programme. To achieve this, an additional keyword Else is used. Syntax:

```
if Condition then Statement1 else Statement2;
```

or

```
if Condition then begin
    Statement;
    Statement;
    …
end
else
    Statement;
    Statement;
    …
end;
```

or

```
if Condition then begin
    Statement;
    Statement;
    …
end
    else Statement2;
```

or

```
if Condition then Statement;
else
    Statement;
    Statement;
    …
end;
```

It is possible to use nested conditional statements. In general terms, statement may be represented by any legitimate instruction of MetaQuotes Language II, except for variable declarations, because, strictly speaking, a variable declaration is not an executable statement.

4.2.4. The WHILE loop

The While loop ensures execution of statements contained in the loop body as many times as the loop condition holds true. A loop may be terminated ahead of schedule using the Break statement. An iteration may be stopped using the Continue statement. This statement causes the start of the next iteration of the loop, i.e. the statements coming after Continue and down to the end of the loop body are not executed in this iteration. It is reasonable to use Break and Continue in a conditional statement. Syntax:

```
while Condition begin
Statement;
[break;][continue;]
```

```
…
end;
```

where Condition is the loop execution condition - a logical expression calculated before each iteration of the loop and taking True or False value; Statement is any instruction of the MetaQuotes Language II. Operator brackets Begin - End denote the loop body and may be replaced by curly brackets { }. Break and Continue statements are not obligatory.
Example:

```
Counter = 1;
while Counter <= Bars begin
print( Close[ Counter - 1 ] );
Counter = Counter + 1;
end;
```

This example illustrates a loop which is going to be executed a Bars times, or, if Bars is 0, not once.

4.2.5. The FOR loop

The For loop ensures execution of statements contained in the loop body a specific number of times. Syntax:

```
for NumberVariable = InitialValue to|downto LimitValue begin
Statement;
[break;][continue;]
…
end;
```

where NumberVariable is the variable of the loop which, after each iteration, will either increase or decrease by one (depending on the use of the keywords To or Downto); InitialValue is the initial numerical value of the loop variable; Statement is any instruction of the MetaQuotes Language II. To or Downto specify an increase of the loop variable by 1 (To) or its decrease by 1(Downto); LimitValue is the marginal numerical value of the loop variable; The operator brackets Begin - End denote the loop body and may be replaced by curly brackets { }. Break and Continue statements are not obligatory. Example:

```
for Counter = 1 to 10 begin
    if Counter > Bars then break;
    print( Close[ Counter ] );
end;
```

This example illustrates a loop which may be executed 10 times. However, if the Bars value is less than 10, the loop breaks earlier, i.e. the loop is executed Bars times.

4.2.6. The BREAK statement

The Break statement ensures an early termination of a For loop or a While loop. The previous example illustrates not only the operation of the loop, but also the use of the Break statement. The Break statement may not be used outside of the loop body. Loops may be nested, and the Break statement terminates only the loop which is closest to it. In other words, a statement breaking an internal loop doesn't break the extertnal loop as well.

4.2.7. The CONTINUE statement

The Continue statement terminates the iteration of the loop ahead of schedule and starts the execution of the next iteration from the beginning of the loop body. In other words, the statements following the Continue are ignored. Example:

```
for Counter = 1 to 10 begin
    …
    if Counter > Bars then continue;
    print( Close[ Counter ] );
    …
end;
```

This example illustrates a loop which will be executed exactly 10 times. However, the value Close[Counter] will be

printed not more than Bars times. The "…" represents other MetaQuotes Language II statements.

4.3. Predefined variables of the trading terminal

For greater convenience of the user, some variables of the trading terminal may be accessed from the Expert Advisor.

AccountNumber
- the account number (synonym AccNum);

Ask
- the Ask price (synonym of the variable PriceAsk);

Balance
- value of the balance of the trading account;

Bars
- number of the bars on the chart - a very important variable showing the degree of filling of the chart with data;

Bid
- the Bid price (synonym of the variable PriceBid);

Close
- the Close price;

Credit
- credit advanced;

Equity
- status of account, including the unrealized profit;

FreeMargin
- value of the Free Margin of the trading account - also used to check the availability of funds on the account;

High
- maximum price over a period;

Low
- minimum price over a period;

Margin
- funds used to support the open positions;

Open
- the Open price;

Point
- value of a single point for the current financial instrument (on which the Expert Advisor runs at the moment), for example, for USD/JPY - 0.01, for USD/CHF - 0.0001 etc;

PriceAsk
- current Ask Price as shown in the "Market Watch" window (synonym of the variable Ask);

PriceBid
- current Bid Price as shown in the "Market Watch" window (synonym of the variable Bid);

PriceHigh
- maximum Ask Price over the current 24-hour period (value from the "Market Watch" window);

PriceLow
- minimum Ask Price over the current 24-hour period (value from the "Market Watch" window);

PriceTime
- current time as shown in the "Market Watch" window;

Time
- bar time if the price chart;

TotalProfit
- total unrealized profit for all the open positions;

TotalTrades
- total number of open positions and delayed orders in the trading terminal;

Volume

- volume (number of the trades over a period).
It should be noted that Close, Open, High, Low, Volume, Time are arrays of historical data (seriesarrays) and allow access of such data over the past periods.

4.4. Built-in functions

MetaQuotes Language II offers a number of functions for the most diverse uses. These include technical indicators [hypertext], trading functions, time functions, mathematical and trigonometric functions, data conversion and output functions, etc.

Abs(nExpression)
- returns the absolute value (module) of the specific numerical value.
Parameter: numerical value.

AccountName
- returns a text string containing the user name (synonym of AccName).

Alert(...)
- produces a dialog screen containing user-defined data.
Any non-zero number of parameters is possible.

ArcCos(nExpression)
- returns a number representing arccosine of nExpression in the range –pi/2 to pi/2 radians.
Parameter: numerical value in the range -1 to 1.

ArcSin(nExpression)
- returns a number representing arcsine of nExpression in the range –pi/2 to pi/2 radians.
Parameter: numerical value in the range -1 to 1.

ArcTan(nExpression)
- returns a number representing arctangent of nExpression in the range –pi/2 to pi/2 radians.
Parameter: numerical value in the range -1 to 1.

Ceil(nExpression)
- returns a number representing the smallest closest integer which is equal or greater than the specified numerical value.
Parameter: numerical value.

CloseOrder(order, lots, price, slippage, color)
- position closing.
Parameters:
order - number of order for the open position;
lots - number of lots;
price - preferred closing price;
slippage - value of the maximum price slippage;
color - color of the cross on the chart.

Comment(...)
- produces user-defines data in the left upper corner of the chart.
Any non-zero number of parameters is possible.

Cos(nExpression)
- calculates and returns the cosine of the numerical value, representing the angle expressed in radians.
Parameter: numerical value.

CurTime
- returns the number of seconds having passed since 0 hours, January 1, 1970.

Day
- returns the sequential number of the current day of the month.

DayOfWeek
- returns the sequential number of the current day of the week. 1 - Sunday, 2 - Monday, ... , 7 - Saturday.

DelArrow(time, price)
- deleting of previously placed pictografic symbol from the chart.
Parameters:
time - time reference point on the chart;
price - price reference point on the chart;
Note. If pass zero as parameters time and price then **all** the pictografic symbols will be removed from the chart. If pass zero as one of parameters (time or price) then such parameter will not be checked and only non-zero parameter will be compared for deleting.

DeleteOrder(order)
- deleting of the previously placed delayed order.
Parameter: order - number of the order of the delayed position.

DelGlobalVariable(name)
- deleting of the specified global variable.
Parameter: name - text string containing global variable name;

DelObject(name, time, price, time_precision, price_precision)
- deleting of the named object created by the MoveObject function.
Parameters :
name - text string with object's name (empty string means any object);
time - one of time reference points on the chart (zero value means any time);
price - one of price reference point on the chart (zero value means any price);
time_precision - deviation from the time value;
price_precision - deviaton from the price value.
Note. If empty name passed then one of possible reference points will be checked in boundaries time-time_precision -- time+time_precision, price-price_precision -- price+price_precision for deleting a some object with any name.

Exp(nExpression)
- returns a number representing the exponent of the specified numerical value.
Parameter: numerical value.

FileDelete(filename)
- deletes a specified file. If the file earlier was opened, then it should be closed, other was deleting will be failed.
Parameter : filename - text string containing a specified file name.

FileClose(filehandle)

- closes a specified file.
Parameter: filehandle - file identification code received from the FileOpen function.

FileOpen(filename, delimiter)
- opens a specified file in the 'Experts' directory and returns the numerical value that identifying opened file (file handle).
Parameters :
filename - text string containing a specified file name. There is 'Comma Separated Values' (CSV) file because data output by FileWrite function is symbolical (text) but not binary and separated with specified delimiter;
delimiter - data delimitation symbol. May be passed as numerical value or as text string containing one symbol only.

FileReadNumber(filehandle)
- returns numerical value read from current position of the specified file. The file pointer is increased by the number of bytes actually read.
Parameter: filehandle - file identification code received from the FileOpen function.

FileReadString(filehandle)
- returns text string read from current position of the specified file. The file pointer is increased by the number of bytes actually read.
Parameter: filehandle - file identification code received from the FileOpen function.

FileSeek(filehandle, offset, origin)
- sets new read/write file position.
Parameters:
filehandle - file identification code received from the FileOpen function.nu
offset - numeric value of the offset relatively of file begin (SEEK_SET), current file position (SEEK_CUR) or end of file (SEEK_END).
origin - one of 3 values - SEEK_SET, SEEK_CUR or SEEK_END.

FileSize(filehandle)
- returns size of the specified file.
Parameter: filehandle - file identification code received from the FileOpen function.

FileTell(filehandle)
- returns current read/write file position.
Parameter: filehandle - file identification code received from the FileOpen function.

FileWrite(filehandle, ...)
- outputs specified data to the specified file into current position. Print function analog. Output data are separated with delimitation symbol specified at file open. After output the file pointer is increased by the number of bytes actually write and in the most of cases passed to the end of file.
Parameters:
filehandle - file identification code received from the FileOpen function.nu
data - comma separated list of any output data.

Floor(nExpression)
- returns a number representing the greatest next integer equal to or smaller than the specified numerical value.
Parameter: numerical value.

GetGlobalVariable(name)

- returns numercal value of defined global variable. If defined global variable does not exist then will return 0 and in the system journal will appear an error message. Parameter: name - text string containing global variable name;

GetIndexValue(shift)
- returns numeric value of the specified element from the primary data array of the current custom indicator. There is specific custom indicator function.
Parameter: shift - shift relative to the current bar (number of periods back), where the data is to be taken from.

GetIndexValue2(shift)
- returns numeric value of the specified element from the secondary data array of the current custom indicator. There is specific custom indicator function.
Parameter: shift - shift relative to the current bar (number of periods back), where the data is to be taken from.

GetTickCount
- operating system's function returns the number of milliseconds that have elapsed since the system was started. Difference between 2 GetTickCount values allows to estimate expert adviser's processing speed.

Highest(type, beginbar, periods)
- returns the shift of the maximum value of Open, Low, High, Close or Volume (depending on the "type" parameter) over the specified number of periods.
Parameters:
type - a returned variable, may take one of the following values: MODE_OPEN, MODE_LOW, MODE_HIGH, MODE_CLOSE, MODE_VOLUME
beginbar - a shift showing the bar, relative to the current bar, that the data should be taken from.
periods - number of periods (in direction from beginbar to the current bar) on which the calculation is carried out.

Hour
- returns the sequential number of the current hour within the 24-hour period.

iAC(shift)
- accelerator-decelerator oscillator.
Parameter: shift - shift relative to the current bar (number of periods back), where the data is to be taken from.

iAD(shift)
- accumulation-distribution indicator.
Parameter: shift - shift relative to the current bar (number of periods back), where the data is to be taken from.

iADX(period, mode, shift)
- index of the average directed motion.
Parameters:
period - number of periods for calculation;
mode - data type may take one of the following values: MODE_MAIN(main indicator), MODE_PLUSDI(line +DI), MODE_MINUSDI(line -DI);
shift - shift relative to the current bar (number of periods back), where the data is to be taken from.

iADXEx(period, applied_price, mode, shift)
- index of the average directed motion (extended function).
Parameters:
period - number of periods for calculation;
applied_price - price to which calculation will be applied: PRICE_CLOSE, PRICE_OPEN, PRICE_HIGH, PRICE_LOW,

PRICE_MEDIAN (median price, (H+L)/2), PRICE_TYPICAL (typical price, (H+L+C)/3), PRICE_WEIGHTED (weighted close, (H+L+C+C)/4) .
mode - data type may take one of the following values: MODE_MAIN(main indicator), MODE_PLUSDI(line +DI), MODE_MINUSDI(line -DI);
shift - shift relative to the current bar (number of periods back), where the data is to be taken from.

iAlligator(jaw_period, jaw_shift, teeth_period, teeth_shift, lips_period, lips_shift, ma_method, applied_price, mode, shift)
- Bill Williams' alligator.
Parameters:
jaw_period - calculating period of the 'gator jaw' line (usually 13);
jaw_shift - forward shift of the 'gator jaw' line (usually 8);
teeth_period - calculating period of the 'gator teeth' line (usually 8);
teeth_shift - forward shift of the 'gator teeth' line (usually 5);
lips_period - calculating period of the 'gator lips' line (usually 5);
lips_shift - forward shift of the 'gator lips' line (usually 3);
ma_method - moving average method may take one of following values: MODE_SMA (simple moving average), MODE_EMA (exponential moving average), MODE_SMMA (smoothed moving average), MODE_LWMA (linear weighted moving average);
applied_price - price to which calculation will be applied: PRICE_CLOSE, PRICE_OPEN, PRICE_HIGH, PRICE_LOW, PRICE_MEDIAN (median price, (H+L)/2), PRICE_TYPICAL (typical price, (H+L+C)/3), PRICE_WEIGHTED (weighted close, (H+L+C+C)/4) .
mode - data source may take one of the following values: MODE_GATORJAW ('blue' balance line), MODE_GATORTEETH ('red' balance line), MODE_GATORLIPS ('green' balance line);
shift - shift relative to the current bar (number of periods back), where the data is to be taken from.

iAO(shift)
- awesome oscillator.
Parameter: shift - shift relative to the current bar (number of periods back), where the data is to be taken from.

iATR(period, shift)
- indicator of the average true interval.
Parameters:
period - number of periods for calculation;
shift - shift relative to the current bar (number of periods back), where the data is to be taken from.

iBands(period, deviation, mode, shift)
- Bollinger band indicator.
Parameters:
period - number of periods for calculation;
deviation - deviation;
mode - may take one of the following values: MODE_MAIN(main line, slipping), MODE_LOW(lower border), MODE_HIGH(upper border);
shift - shift relative to the current bar (number of periods back), where the data is to be taken from.

iBandsEx(period, deviation, bands_shift, applied_price, mode, shift)
- Bollinger band indicator (extended function).
Parameters:
period - number of periods for calculation;
deviation - deviation;

bands_shift - forward or backward shift for calculated values;
applied_price - price to which calculation will be applied: PRICE_CLOSE, PRICE_OPEN, PRICE_HIGH, PRICE_LOW, PRICE_MEDIAN (median price, (H+L)/2), PRICE_TYPICAL (typical price, (H+L+C)/3), PRICE_WEIGHTED (weighted close, (H+L+C+C)/4) .
mode - may take one of the following values: MODE_MAIN(main line, slipping), MODE_LOW(lower border), MODE_HIGH(upper border);
shift - shift relative to the current bar (number of periods back), where the data is to be taken from.

iBearsPower(period, applied_price, shift)
- bears power indicator.
Parameters:
period - number of periods for calculation;
applied_price - price to which calculation will be applied: PRICE_CLOSE, PRICE_OPEN, PRICE_HIGH, PRICE_LOW, PRICE_MEDIAN (median price, (H+L)/2), PRICE_TYPICAL (typical price, (H+L+C)/3), PRICE_WEIGHTED (weighted close, (H+L+C+C)/4) .
shift - shift relative to the current bar (number of periods back), where the data is to be taken from.

iBullsPower(period, applied_price, shift)
- bulls power indicator.
Parameters:
period - number of periods for calculation;
applied_price - price to which calculation will be applied: PRICE_CLOSE, PRICE_OPEN, PRICE_HIGH, PRICE_LOW, PRICE_MEDIAN (median price, (H+L)/2), PRICE_TYPICAL (typical price, (H+L+C)/3), PRICE_WEIGHTED (weighted close, (H+L+C+C)/4) .
shift - shift relative to the current bar (number of periods back), where the data is to be taken from.

iBWMFI(shift)
- market facilitation index.
shift - shift relative to the current bar (number of periods back), where the data is to be taken from.

iCCI(period, shift)
- commodity channel index.
Parameters:
period - number of periods for calculation;
shift - shift relative to the current bar (number of periods back), where the data is to be taken from.

iCCIEx period, applied_price, shift)
- commodity channel index (extended function).
Parameters:
period - number of periods for calculation;
applied_price - price to which calculation will be applied: PRICE_CLOSE, PRICE_OPEN, PRICE_HIGH, PRICE_LOW, PRICE_MEDIAN (median price, (H+L)/2), PRICE_TYPICAL (typical price, (H+L+C)/3), PRICE_WEIGHTED (weighted close, (H+L+C+C)/4) .
shift - shift relative to the current bar (number of periods back), where the data is to be taken from.

iCustom
- custom indicator.
Syntax: iCustom(name [, parameter1 [,parameter2 ....]], mode, shift)
Parameters:
name - text string containing custom indicator program name;

parameter1, parameter2... - numeric input parameters (if needed).
mode - may take one of the following values: MODE_FIRST (value from the primary data array), MODE_SECOND (value from the secondary data array).
shift - shift relative to the current bar (number of periods back), where the data is to be taken from.

Custom indicator program is a simple expert adviser written on the MQLII and passed translation (verify button in the MetaEditor) Parameters for this program are passed in the same sequence as they are listed in the 'Inputs' section. Only numerical values can be passed as parameters! For the preparation of the primary and secondary (if needed) data arrays of the custom indicator are to be used SetIndexValue and SetIndexValue2 functions. It is impossible to use some trade functions, custom indicator or user function call functions in the custom indicator.

iDeMarker(period, shift)
- DeMark indicator.
Parameters:
period - number of periods for calculation;
shift - shift relative to the current bar (number of periods back), where the data is to be taken from.

iEnvelopes(period, ma_method, deviation, applied_price, mode, shift)
- envelopes indicator.
Parameters:
period - number of periods for calculation;
ma_method - moving average method may take one of following values: MODE_SMA (simple moving average), MODE_EMA (exponential moving average), MODE_SMMA (smoothed moving average), MODE_LWMA (linear weighted moving average);
deviation - deviation percent;
applied_price - price to which calculation will be applied: PRICE_CLOSE, PRICE_OPEN, PRICE_HIGH, PRICE_LOW, PRICE_MEDIAN (median price, (H+L)/2), PRICE_TYPICAL (typical price, (H+L+C)/3), PRICE_WEIGHTED (weighted close, (H+L+C+C)/4) .
mode - can take one of values: MODE_UPPER (upper band value), MODE_LOWER (lower band value).
shift - shift relative to the current bar (number of periods back), where the data is to be taken from.

iForce(period, ma_method, applied_price, shift)
- force indicator.
Parameters:
period - number of periods for calculation;
ma_method - moving average method may take one of following values: MODE_SMA (simple moving average), MODE_EMA (exponential moving average), MODE_SMMA (smoothed moving average), MODE_LWMA (linear weighted moving average);
applied_price - price to which calculation will be applied: PRICE_CLOSE, PRICE_OPEN, PRICE_HIGH, PRICE_LOW, PRICE_MEDIAN (median price, (H+L)/2), PRICE_TYPICAL (typical price, (H+L+C)/3), PRICE_WEIGHTED (weighted close, (H+L+C+C)/4) .
shift - shift relative to the current bar (number of periods back), where the data is to be taken from.

iFractals(mode, shift)
- fractals. If zero value returned then no any fractal presented.
Parameters:
mode - can take one of values: MODE_UPPER (upper line values), MODE_LOWER (lover line values).
shift - shift relative to the current bar (number of periods back), where the data is to be taken from.

iGator(jaw_period, jaw_shift, teeth_period, teeth_shift, lips_period, lips_shift, ma_method, applied_price, mode,

shift)
- gator oscillator.
Parameters:
jaw_period - calculating period of the 'gator jaw' line (usually 13);
jaw_shift - forward shift of the 'gator jaw' line (usually 8);
teeth_period - calculating period of the 'gator teeth' line (usually 8);
teeth_shift - forward shift of the 'gator teeth' line (usually 5);
lips_period - calculating period of the 'gator lips' line (usually 5);
lips_shift - forward shift of the 'gator lips' line (usually 3);
ma_method - moving average method may take one of following values: MODE_SMA (simple moving average), MODE_EMA (exponential moving average), MODE_SMMA (smoothed moving average), MODE_LWMA (linear weighted moving average);
applied_price - price to which calculation will be applied: PRICE_CLOSE, PRICE_OPEN, PRICE_HIGH, PRICE_LOW, PRICE_MEDIAN (median price, (H+L)/2), PRICE_TYPICAL (typical price, (H+L+C)/3), PRICE_WEIGHTED (weighted close, (H+L+C+C)/4) .
mode - data source may take one of the following values: MODE_UPPER (absolute difference between 'blue' and 'red' lines), MODE_LOWER (negative absolute difference between 'red' and 'green' lines);
shift - shift relative to the current bar (number of periods back), where the data is to be taken from.

iIchimoku(tenkan_sen, kijun_sen, senkou_span_b, mode, shift)
- Ichimoku Kinko Hyo.
Parameters:
tenkan_sen - Tenkan Sen period (usually 9);
kijun_sen - Kijun Sen period (usually 26);
senkou_span_b - Senkou SpanB period (usually 52);
mode - data source may take one of the following values: MODE_TENKANSEN, MODE_KIJUNSEN, MODE_SENKOUSPANA, MODE_SENKOUSPANB, MODE_CHINKOUSPAN;
shift - shift relative to the current bar (number of periods back), where the data is to be taken from.

iMA(period, mode, shift)
- moving average indicator.
Parameters:
period - number of periods for calculation;
mode - mode of calculation, which may take one of the following values: MODE_SMA, MODE_EMA, MODE_WMA.
shift - shift relative to the current bar (number of periods back), where the data is to be taken from.

iMAEx(period, ma_shift, applied_price, ma_method, shift)
- moving average indicator (extended function).
Parameters:
period - number of periods for calculation;
ma_shift - forward or backward shift of the calculated values;
applied_price - price to which calculation will be applied: PRICE_CLOSE, PRICE_OPEN, PRICE_HIGH, PRICE_LOW, PRICE_MEDIAN (median price, (H+L)/2), PRICE_TYPICAL (typical price, (H+L+C)/3), PRICE_WEIGHTED (weighted close, (H+L+C+C)/4) .
ma_method - moving average method may take one of following values: MODE_SMA (simple moving average), MODE_EMA (exponential moving average), MODE_SMMA (smoothed moving average), MODE_LWMA (linear weighted moving average);
shift - shift relative to the current bar (number of periods back), where the data is to be taken from.

iMACD(fast_ema_period, slow_ema_period, signal_period, mode, shift)

- moving averages convergence/divergence indicator.
Parameters:
fast_ema_period - number of periods for calculation of the 'fast' moving average (usually 12);
slow_ema_period - number of periods for calculation of the 'slow' moving average (usually 26);
signal_period - number of periods for calculation of the 'signal' moving average (usually 9);
mode - source of data, may take one of the following values: MODE_MAIN (main indicator), MODE_SIGNAL (signal line);
shift - shift relative to the current bar (number of periods back), where the data is to be taken from.

iMACDEx(fast_ema_period, slow_ema_period, signal_period, applied_price, mode, shift)
- moving averages convergence/divergence indicator (extended function).
Parameters:
fast_ema_period - number of periods for calculation of the 'fast' moving average (usually 12);
slow_ema_period - number of periods for calculation of the 'slow' moving average (usually 26);
signal_period - number of periods for calculation of the 'signal' moving average (usually 9);
applied_price - price to which calculation will be applied: PRICE_CLOSE, PRICE_OPEN, PRICE_HIGH, PRICE_LOW, PRICE_MEDIAN (median price, (H+L)/2), PRICE_TYPICAL (typical price, (H+L+C)/3), PRICE_WEIGHTED (weighted close, (H+L+C+C)/4) .
mode - source of data, may take one of the following values: MODE_MAIN (main indicator), MODE_SIGNAL (signal line);
shift - shift relative to the current bar (number of periods back), where the data is to be taken from.

iMFI(period, shift)
- money flow index.
Parameters:
period - number of periods for calculation;
shift - shift relative to the current bar (number of periods back), where the data is to be taken from.

iMom(period, shift)
- momentum indicator.
Parameters:
period - number of periods for calculation;
shift - shift relative to the current bar (number of periods back), where the data is to be taken from.

iMomEx(period, applied_price, shift)
- momentum indicator (extended function).
Parameters:
period - number of periods for calculation;
applied_price - price to which calculation will be applied: PRICE_CLOSE, PRICE_OPEN, PRICE_HIGH, PRICE_LOW, PRICE_MEDIAN (median price, (H+L)/2), PRICE_TYPICAL (typical price, (H+L+C)/3), PRICE_WEIGHTED (weighted close, (H+L+C+C)/4) .
shift - shift relative to the current bar (number of periods back), where the data is to be taken from.

iOBV(shift)
- on balance volume indicator.
Parameter: shift - shift relative to the current bar (number of periods back), where the data is to be taken from.

iOsMA(fast_ema_period, slow_ema_period, signal_period, applied_price, shift)
- moving averages of oscillator.
Parameters:

fast_ema_period - number of periods for calculation of the 'fast' moving average (usually 12);
slow_ema_period - number of periods for calculation of the 'slow' moving average (usually 26);
signal_period - number of periods for calculation of the 'signal' moving average (usually 9);
applied_price - price to which calculation will be applied: PRICE_CLOSE, PRICE_OPEN, PRICE_HIGH, PRICE_LOW, PRICE_MEDIAN (median price, (H+L)/2), PRICE_TYPICAL (typical price, (H+L+C)/3), PRICE_WEIGHTED (weighted close, (H+L+C+C)/4) .
shift - shift relative to the current bar (number of periods back), where the data is to be taken from.

iRSI(period, shift)
- relative strenghth index.
Parameters:
period - number of periods for calculation;
shift - shift relative to the current bar (number of periods back), where the data is to be taken from.

iRSIEx(period, period2, applied_price, mode, shift)
- relative strenghth index (extended function).
Parameters:
period - number of periods for calculation;
period2 - number of periods for calculation of the second line;
applied_price - price to which calculation will be applied: PRICE_CLOSE, PRICE_OPEN, PRICE_HIGH, PRICE_LOW, PRICE_MEDIAN (median price, (H+L)/2), PRICE_TYPICAL (typical price, (H+L+C)/3), PRICE_WEIGHTED (weighted close, (H+L+C+C)/4) .
mode - source of data, may take one of the following values: MODE_FIRST (main indicator), MODE_SECOND (second line);
shift - shift relative to the current bar (number of periods back), where the data is to be taken from.

iRVI(period, mode, shift)
- relative vigor index.
Parameters:
period - number of periods for calculation;
mode - source of data, may take one of the following values: MODE_MAIN (main indicator), MODE_SIGNAL (signal line);
shift - shift relative to the current bar (number of periods back), where the data is to be taken from.

iSAR(step, maximum, shift)
- parabolic SAR.
Parameters:
step - increment, usually 0.02;
maximum - maximum value, usually 0.2.

iSTO(%Kperiod, %Dperiod, slowing, method, mode, shift)
- "stochastic oscillator" indicator.
Parameters:
%Kperiod - %K line period;
%Dperiod - %D line period;
slowing - slowing value;
method - method of calculation, may take one of the following values: MODE_SMA (simple average), MODE_EMA (exponential), MODE_WMA (weighted);
mode - source of data, may take one of the following values: MODE_MAIN (main indicator line), MODE_SIGNAL (signal indicator line);

shift - shift relative to the current bar (number of periods back), where the data is to be taken from.

iWPR(period, shift)
- Williams percentage range indicator.
Parameters:
period - number of periods for calculation;
shift - shift relative to the current bar (number of periods back), where the data is to be taken from.

IsDemo
- returns logical value True if the Expert Advisor runs on a traning account. Otherwise returns False value.

IsFileEnded(filehandle)
- returns logical value 'True' if end of the specified file during the read reached..
Parameter: filehandle - file identification code received from the FileOpen function.

IsFileLineEnded(filehandle)
- returns logical value 'True' if end of line ('carriage return' or 'line feed') symbol during the read reached.
Parameter: filehandle - file identification code received from the FileOpen function.

IsGlobalVariable(name)
- returns logical value 'True' if the specified global variable exists. Or 'False' otherwise.
Parameter: name - text string containing global variable name;

IsIndirect
- returns logical value 'True' if the specified instrument is calculated using the reverse method. Otherwise returns False value.

IsTesting
- returns logical value 'True' if expert advisor launched for testing. Or 'False' otherwise.

IsTradeAllowed
- returns logical value 'True' if checkbox "allow live trading" in expert advisor's properties is checked. Or 'False' otherwise.

LastTradeTime
- returns a number representing the time of execution of the most recent trade (SetOrder, DelOrder, CloseOrder, ModifyOrder) in seconds having passed since 0 hours, January 1, 1970.

Log(nExpression)
- returns a logarithm of the specified positive numerical value.
Parameter: positive numerical value.

Lowest(type, beginbar, periods)
- returns the shift of the least value of Open, Low, High, Close or Volume (depending on the "type" parameter) over a specific number of periods.
Parameters:
type - may take one of the following values: MODE_OPEN, MODE_LOW, MODE_HIGH, MODE_CLOSE, MODE_VOLUME
beginbar - a shift showing the bar, relative to the current bar, that the data should be taken from.
periods - number of periods (in direction from beginbar to the current bar) on which the calculation is carried out.

MarketInfo(symbol, mode)
- returns one of specified values from the MarketWatch window.
Parameters:
symbol - the financial instrument short name of which information is needed;
mode - type of returned data can take one of the values: MODE_HIGH, MODE_LOW, MODE_BID, MODE_ASK, MODE_TIME.

Max(nExpression1, nExpression2)
- returns maximal value of two specified numeric values.

Min(nExpression1, nExpression2)
- returns minimal value of two specified numeric values.

Minute
- returns the sequential number of the current minute in the hour.

Mod(nExpression1, nExpression2)
- returns an integer representing the remainder of division of one numerical value by another one.
Parameters: numerical value1, numerical value2.

ModifyOrder(order, price, stoploss, takeprofit, color)
- modification of characteristics for the previously opened position or a delayed order.
Parameters:
order - number or order of the opened or delayed position;
price - new price (only for a pending orders!);
stoploss - new stop loss level;
takeprofit - new profit-taking level;
color - color of the pictogram on the chart.

Month
- returns the sequential number of the current month.

MoveObject(name, type, time, price, time2, price2 [,color [,weight [, style]]])
- moving or creation of a named object.
Parameters:
name - name of the object in a text string form;
type - object type, may take one of the following values: OBJ_HLINE (horizontal line), OBJ_VLINE (vertical line), OBJ_TRENDLINE (trend line), OBJ_SYMBOL (setting a pictogram), OBJ_TEXT (text string), OBJ_FIBO (Fibonacchi retracement);
time - first chart reference point by time;
price - first chart reference point by price;
time2 - second chart reference point by time;
price2 - second chart reference point by price
Optional parameters:
color - object's color;
weight - line wight in pixels (values from 1 to 5 are allowed);
style - line style, may take one of values: STYLE_SOLID, STYLE_DASH, STYLE_DOT, STYLE_DASHDOT, STYLE_DASHDOTDOT.
Note: both reference points are used for trend lines and Fibonacchi retracement. For text string or for pictografical symbol is used second reference point only, first reference point is ignored. For any horizontal line is used 'price2' value

only. For vertical line - 'time2' value.

Normalize(number, precision)
- returns numeric value rounded to the specified precision.
Parameters:
number - specified numerical value;
precision - precision format, number of digits after the decimal point.

NumberToStr(number, precision)
- returns text string with the specified numerical value transformed into the specified precision format. The function used to produce numerical values with other precision format than 4 digits after the decimal point.
Parameters:
number - specified numerical value;
precision - precision format, number of digits after the decimal point.

OrderValue(position, mode)
- returns one of the specified values of the order.
Parameters:
position - position of the order in the trading terminal list, starting from 1;
mode - type of the returned data, may take one of the following values: VAL_TICKET (order number), VAL_OPENTIME (order opening time), VAL_TYPE (order type), VAL_LOTS (number of requested lots), VAL_SYMBOL (name of the instrument, in form of a text string), VAL_OPENPRICE (opening price), VAL_STOPLOSS (stop loss level), VAL_TAKEPROFIT (take profit level), VAL_CLOSEPRICE (closing price), VAL_COMISSION (commission amount), VAL_SWAP (amount of rollover fees for the position rollover), VAL_PROFIT (amount of profit of the trade), VAL_COMMENT (comment on the particular position in form of a text string), VAL_CLOSETIME (order closing time).

Ord
- same as in OrderValue.

Period
- returns the number of minutes defining the used period.

Pow(nBaseExpression, nExponentExpression)
- returns the number produced by raising the nBaseExpression number to the nExponentExpression power, given as parameters.
Parameters: numerical value1, numerical value2.

Print(...)
- prints the data defined by the user into the system log.
Any non-zero number of parameters possible.

PrintTrade(position)
- stores the details of a particular position in the log.
Parameter: position - number of an opened position in the trading terminal.

Rand
- returns a generated pseudorandom number.
Before using this function, pseudorandom number generator should be set to initial position using the Srand function. If the pseudorandom number generator is to be used, it is set into the initial position once, at the start of the programme.

Round(nExpression)
- returns a number representing a specified numerical value rounded to the closest integer.
Parameter: a numerical value.

Seconds
- returns the sequential number of the current second in the minute.

ServerAddress
- returns the server IP-address in form of a text string.

SetArrow(time, price, symbol, color)
- setting the pictographic symbol on the chart.
Parameters:
time - chart reference point by time;
price - chart reference point by price;
symbol - numerical value of the symbol from the Wingdings font set;
color - pictogram color.

SetDebugMode(mode)
- sets debug info output mode.
Parameter: mode - debug info output mode can take one of values: MODE_FILE (output to file; all the debug info as well as printed data will output into the separate log-file EXPERT_NAME.log), MODE_VALUES (output all the variables values after expert advisor processing), MODE_TIME (output expert advisor's elapsed time, executed instructions count, remaining stack size), 0 (al the set modes cancellation); Modes can be combined one with other with the adding it's values (for example: MODE_FILE + MODE_VALUES + MODE_TIME).
Note. This function still not working.

SetGlobalVariable(name, value)
- sets specified numerical value to the specified global variable. If the specified global variable does not exist then it will be created. Global variables are accessible to any expert advisor. Global variables are kept in the 'gvariables.dat' file. If some global variable will not used in the period of 2 weeks then this one will be deleted automatically.
Parameters:
name - text string containing global variable name;
value - specified numerical value.

SetIndexValue(shift, value)
- sets specified numeric value to the specified primary data array's element of the current custom indicator. There is custom indicator program' specific function.
Parameters:
shift - shift relative to the current bar (number of periods back), where the data is to be taken from.
value - numeric value to set.

SetIndexValue2(shift, value)
- sets specified numeric value to the specified secondary data array's element of the current custom indicator. There is custom indicator program' specific function.
Parameters:
shift - shift relative to the current bar (number of periods back), where the data is to be taken from.
value - numeric value to set.

SetLoopCount count)

- sets limitation quantity of executed instructions. This function allows to limit the performance time of the expert advisor. Value 1000000 is used by default. Special value 0 allows to turn off the control of the quantity of executed instructions, however in purposes of an infinite loop protection the expert advisor will not work more than one second.
Parameter: count - limitation quantity of executed instructions.

SetObjectText(name, time, price, symbol, color)
- assign a text string to a specified object of OBJ_TEXT type.
Parameters:
name - object name;
text - specified text;
font - font name;
size - font size;
color - text color.

SetOrder(operation, lots, price, slippage, stoploss, takeprofit, color)
- main function used to open a position or set a delayed order.
Parameters:
operation - type of operation, may take the following values: OP_BUY (opening the buying position), OP_SELL (opening the selling position), OP_BUYLIMIT, OP_SELLLIMIT, OP_BUYSTOP, OP_SELLSTOP (putting the delayed order);
lots - number of lots;
price - preferred closing price of the trade;
slippage - maximum price slippage for OP_BUY and OP_SELL operations;
stoploss - Stop Loss level;
takeprofit - Profit Taking level;
color - color of the arrow put on the chart when the function is called.

SetText(time, price, string, color)
- put a text string on the chart in the specified position.
Parameters:
time - chart reference point by time;
price - chart reference point by price;
string - text string;
color - text color.

Sin(nExpression)
- calculates and returns the sine of the numerical value representing the angle in radians.
Parameter: numerical value.

Sqrt(nExpression)
- calculates and returns the square root of the specified positive numerical value.
Parameter: positive numerical value.

Srand(initial_value)
- sets the pseudorandom number generator to the initial position. If the generator is to be used, it is set into the initial position once, at the start of the programme. The best value to be used for initial setting is the number returned by the Time function - in this case the randomness of the sequence generation is increased.
Parameter: positive numerical value.

StrToTime(DateTimeString)
- returns numeric time value, converted from text string such as "yyyy.mm.dd hh:mi"

Parameter: date and(or) time as string.
Note: strings as "yyyy.mm.dd hh:mi", "yyyy.mm.dd", "hh:mi" are allowed. Missing data (date or time) assumes current date or current time.

Symbol
- returns a text string with the name of the financial instrument the Expert Advisor is running on.

Tan(nExpression)
- calculates and returns the tangent of a numerical value representing an angle in radians. Parameter: numerical value.

TimeDay(Time)
- returns sequential day number in the month of the specified time value.
Parameter: positive numerical value.

TimeDayOfWeek(Time)
- returns sequential day number in the week of the specified time value. 1 - sunday, 2 - monday, … , 7 - saturday.
Parameter: positive numerical value.

TimeHour(Time)
- returns sequential hour (from 0 to 23) number in the day of the specified time value.
Parameter: positive numerical value.

TimeMinute(Time)
- returns sequential minute (from 0 to 59) number in the hour of the specified time value.
Parameter: positive numerical value.

TimeMonth(Time)
- returns sequential month number in the year of the specified time value.
Parameter: positive numerical value.

TimeSeconds(Time)
- returns sequential second (from 0 to 59) number in the minute of the specified time value.
Parameter: positive numerical value.

TimeToStr(Time)
- returns text string of "yyyy.mm.dd hh:mi" type, produced from the specified numerical value representing the number of seconds having passed since 0 hours, January 1, 1970.

Parameter: positive numerical value.

TimeYear(Time)
- returns the number of the year of the specified time value.
Parameter: positive numerical value.

UserFunction
- calls for execution specified MQLII-program(user function).
Syntax: UserFunction(name [, parameter1 [,parameter2 ....]])
Параметры:
name - user function name;
parameter1, parameter2... - numeric parameters (if needed).

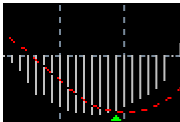
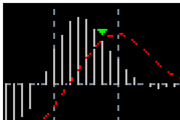
User function program is a simple expert adviser written on the MQLII and passed translation (verify button in the MetaEditor) Parameters for this program are passed in the same sequence as they are listed in the 'Inputs' section. Only numerical values can be passed as parameters! For the returning of some value is to be used Return function. UserFunction function returns the same value that was set by the Return function in the user function program. Return function is specific for the user function program.

Year
- returns the number of the current year.

Step-By-Step Creation of Simple Expert

Let's try to create an Expert Advisor running on the standard MACD indicator, with the profit-taking capability, supporting trailing stops and using most safeguards for secure operation. In the example given below the trade is done by opening and controlling a single position.

Trading principles:

- **Long (BUY) entry** - MACD indicator is below zero, goes upwards and is crossed by the Signal Line going downwards.

- **Short (SELL) entry** - MACD indicator is above zero, goes downwards and is crossed by the Signal Line going upwards.

- **Long exit** - by execution of the take profit limit, by execution of the trailing stop or when MACD crosses its Signal Line (MACD is above zero, goes downwards and is crossed by the Signal Line going upwards).
- **Short exit** - by execution of the take profit limit, by execution of the trailing stop or when MACD crosses its Signal Line (MACD is below zero, goes upwards and is crossed by the Signal Line going downwards).

Important note: to exclude insignificant changes of the MACD indicator (small 'hillocks' on the chart) from our analysis, we introduce an additional measure of controlling the size of the plotted 'hillocks' as follows: the indicator size should be at least 5 units of the minimum price (5*Point, which for USD/CHF = 0.0005 and for USD/JPY = 0.05).

Step I: Writing the Expert Advisor description

Point the mouse pointer on the **Expert Advisors** section of the **Navigator** window, press the right button of the mouse and select "Create a new Expert" command in the appearing menu. The Initializing Wisard of the Expert Advisor will ask you to enter certain data. In the appearing window we write the name (Name) of the Expert Advisor - MACD Sample, the author (Author) - indicate your name, the link (Link) - a link to your website, in the notes (Notes) - Test example of an MACD-based Expert Advisor.

Step II: Creating the primary structure of the programme

Code of the test Expert Advisor will only occupy several pages, but even such volume is often difficult to grasp, especially in view of the fact that we are not professional programmers - otherwise we wouldn't need this description at all, right? :)

To get some idea of the structure of a standard Expert Advisor, let's take a look at the description given below:

1. Initial data checks
   - check the chart, number of bars on the chart
   - check the values of external variables: Lots, S/L, T/P, T/S
2. Setting the internal variables for quick data access
3. Checking the trading terminal - is it void? If yes, then:
   - checks: availability of funds on the account etc...

    - is it possible to take a long position (BUY)?
        - open a long position and exit
    - is it possible to take a short position (SELL)?
        - open a short position and exit

    exiting the Expert Advisor...
4. Control of the positions previously opened in the cycle
    - if it is a long position
        - should it be closed?
        - should the trailing stop be reset?
    - if it is a short position
        - should it be closed?
        - should the trailing stop be reset?

It turned out to be relatively simple, with only 4 main blocks.
Now let's try to generate pieces of code step by step for each section of the structural scheme:

1. **Initial data checks**
   This piece of code usually migrates from one Expert Advisor to another with minor changes - it is a practically standard block of checks:

```
If Bars<200 Then Exit;      // the chart has less than 200 bars - exit
If TakeProfit<10 Then Exit; // wrong takeprofit parameters
```

2. **Setting the internal variables for quick data access**
   In the programme code it is very often necessary to access the indicator values or handle the computed values. To simplify the coding and speed up the access, the data is initially nested within the internal variables.

```
MacdCurrent=iMACD(12,26,9,0,MODE_MAIN);     // MACD value on the current bar
MacdPrevious=iMACD(12,26,9,1,MODE_MAIN);    // MACD value on the previous bar
SignalCurrent=iMACD(12,26,9,0,MODE_SIGNAL); // Signal Line value on the current bar
SignalPrevious=iMACD(12,26,9,1,MODE_SIGNAL);// Signal Line value on the previous bar
MaCurrent=iMA(MATrendPeriod,MODE_EMA,0);    // moving average value on the current bar
MaPrevious=iMA(MATrendPeriod,MODE_EMA,1);   // moving average value on the previous bar
```

   Now, instead of the monstrous notation of **iMACD(12,26,9,MODE_MAIN,0)** we can simply write in the programme text **MacdCurrent**. All the variables used by the Expert Advisor will have to be preliminarily described, according to the MetaQuotes Language II description. Therefore we insert the description of these variables at the beginning of the programme:

```
var: MacdCurrent(0), MacdPrevious(0), SignalCurrent(0), SignalPrevious(0);
var: MaCurrent(0), MaPrevious(0);
```

   MetaQuotes Language II also introduces the concept of additional user-defined variables which may be set from outside the programme, without any interference with the source source text of the Expert Advisor programme. This feature allows added flexibility. Variable **MATrendPeriod** is a user-defined variable of this very type. So, we insert the description of this variable in the beginning of the programme.

```
defines: MATrendPeriod(56);
```

3. **Checking the trading terminal - is it void? If yes, then:**
   In our Expert Advisor we only use the current positions and don't handle the delayed orders. However, to be on the safe side, let's introduce a check of the trading terminal for previously placed orders:

```
If TotalTrades < 1 then   // no opened orders identified
  {
```

   - **checks: availability of funds on the account etc...**
     Before analyzing the market situation it is advisable to check the status of your account to make sure that there are free funds for opening a position.

     ```
     If FreeMargin < 1000 then Exit;  // no funds - exit
     ```

   - **is it possible to take a long position (BUY)?**
     Condition of entry into the long position: MACD is below zero, goes upwards and is crossed by the Signal Line going downwards. This is how we describe it in MQLII (note that we operate on the indicator values which were previously saved in the variables):

     ```
     If MacdCurrent < 0 and MacdCurrent > SignalCurrent and
        MacdPrevious < SignalPrevious and              // a cross-section exists
        Abs(MacdCurrent) > (MACDOpenLevel*Point) and   // the indicator plotted a decent 'hill
        MaCurrent > MaPrevious then                    // 'bull' trend
       {
        SetOrder(OP_BUY,Lots,Ask,3,0,Ask+TakeProfit*Point,RED); // executing
        Exit;    // exiting, since after the execution of a trade
                 // there is a 10-second trading timeout
       };
     ```

     Above we mentioned the method of additional monitoring of the size of the plotted 'hillocks'. MACDOpenLevel variable is a user-defined variable which may be changed without interfering with the programme text, to ensure greater flexibility. In the beginning of the programme we insert a description of this variable (as well as the description of the variable used below).

     ```
     defines: MACDOpenLevel(3), MACDCloseLevel(2);
     ```

   - **is it possible to take a short position (SELL)?**
     Condition of entry of a short position: MACD is above zero, goes downwards and is crossed by the Signal Line going upwards. The notation is as follows:

     ```
     If MacdCurrent > 0 and MacdCurrent < SignalCurrent and
        MacdPrevious > SignalPrevious and MacdCurrent > (MACDOpenLevel*Point) and
        MaCurrent < MaPrevious then
       {
        SetOrder(OP_SELL,Lots,Bid,3,0,Bid-TakeProfit*Point,RED); // executing
        Exit; // exiting
       };
     ```

```
   Exit;   // no new positions opened - just exit
  };
```

4. **Control of the positions previously opened in the cycle**

```
for cnt=1 to TotalTrades
  {
   if OrderValue(cnt,VAL_TYPE)<=OP_SELL and   // is it an open position?
      OrderValue(cnt,VAL_SYMBOL)=Symbol then  // position from "our" chart?
     {
```

Cnt is the cycle variable which is to be described at the beginning of the programme as follows:
var: Cnt(0);

- if it is a long position

```
If OrderValue(cnt,VAL_TYPE)=OP_BUY then // long position opened
  {
```

  - **should it be closed?**
    Condition for exiting a long position: MACD is crossed by its Signal Line, MACD being above zero, going downwards and being crossed by the Signal Line going upwards.

```
If MacdCurrent > 0 and MacdCurrent < SignalCurrent and
   MacdPrevious > SignalPrevious and MacdCurrent > (MACDCloseLevel*Point) then
  {
   CloseOrder(OrderValue(cnt,VAL_TICKET),OrderValue(cnt,VAL_LOTS),Bid,3,Violet);
   Exit; // exit
  };
```

  - **should the trailing stop be reset?**
    We set the trailing stop only in case the position already has a profit exceeding the trailing stop level in points, and in case the new level of the stop is better than the previous.

```
If TrailingStop > 0 then // if trailing stops are used
  {
   If (Bid-OrderValue(cnt,VAL_OPENPRICE))>(Point*TrailingStop) then
     {
      If OrderValue(cnt,VAL_STOPLOSS)<(Bid-Point*TrailingStop) then
        {
         ModifyOrder(OrderValue(cnt,VAL_TICKET),OrderValue(cnt,VAL_OPENPRICE),
                     Bid-Point*TrailingStop,OrderValue(cnt,VAL_TAKEPROFIT),Red)
         Exit;
        };
     };
  };

}
```

- **if it is a short position**

```
else // otherwise it is a short position
  {
```

- **should it be closed?**
  Condition for exiting a short position: MACD is crossed by its Signal Line, MACD being below zero, going upwards and being crossed by the Signal Line going downwards.

```
If MacdCurrent<0 and MacdCurrent>SignalCurrent and
   MacdPrevious(MACDCloseLevel*Point) then
  {
   CloseOrder(OrderValue(cnt,VAL_TICKET),OrderValue(cnt,VAL_LOTS),Ask,3,Violet);
   Exit; // exit
  };
```

- **should the trailing stop be reset?**
  We set the trailing stop only in case the position already has a profit exceeding the trailing stop level in points, and in case the new level of the stop is better than the previous.

```
If TrailingStop>0 then  // the user has put a trailing stop in his settings
  {                // so, we set out to check it
    If (OrderValue(cnt,VAL_OPENPRICE)-Ask)>(Point*TrailingStop) then
      {
        If OrderValue(cnt,VAL_STOPLOSS)=0 or
           OrderValue(cnt,VAL_STOPLOSS)>(Ask+Point*TrailingStop) then
          {
           ModifyOrder(OrderValue(cnt,VAL_TICKET),OrderValue(cnt,VAL_OPENPRICE),
                       Ask+Point*TrailingStop,OrderValue(cnt,VAL_TAKEPROFIT),Red);
           Exit;
          };
      };
  };
```

// end. Closing all the curly bracket which remain open.

```
    };
  };
};
```

So, following this step-by-step procedure, we have written our Expert Advisor...

Step III: Assembling the resulting code of the programme

Let's open the Expert Advisor settings (using a button or a line in the Properties... menu. We are offered a window in which we have to define the external settings of the working parameters:

Let's assemble all the code from the previous section:

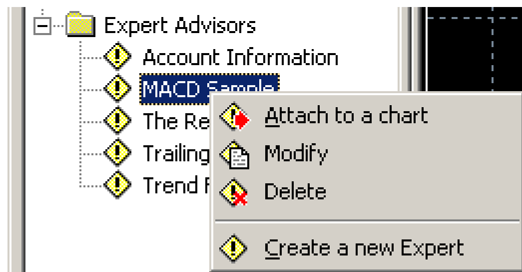
```
defines: MACDOpenLevel(3),MACDCloseLevel(2);
defines: MATrendPeriod(56);
var:     MacdCurrent(0),MacdPrevious(0),SignalCurrent(0),SignalPrevious(0);
var:     MaCurrent(0),MaPrevious(0);
var:     cnt(0);

// initial data checks
// it is important to make sure that the Expert Advisor runs on a normal chart and that
// the user has correctly set the external variables (Lots, StopLoss,// TakeProfit, TrailingS
// in our case we only check the TakeProfit
If Bars<200 or TakeProfit<10 then Exit;  // less than 200 bars on the chart

// to simplify and speed up the procedure, we store the necessary
// indicator data in temporary variables
MacdCurrent   =iMACD(12,26,9,0,MODE_MAIN);
MacdPrevious  =iMACD(12,26,9,1,MODE_MAIN);
SignalCurrent =iMACD(12,26,9,0,MODE_SIGNAL);
SignalPrevious=iMACD(12,26,9,1,MODE_SIGNAL);
MaCurrent     =iMA(MATrendPeriod,MODE_EMA,0);
MaPrevious    =iMA(MATrendPeriod,MODE_EMA,1);

// now we have to check the status of the trading terminal.
// we are going to see whether there are any previously opened positions or orders.
If TotalTrades<1 then
  {
   // there are no opened orders
   // just to be on the safe side, we make sure we have free funds on our account.
   // the "1000" value is taken just as an example, usually it is possible to open 1 lot
   If FreeMargin<1000 then Exit;  // no money - we exit

   // checking for the possibility to take a long position (BUY)
   If MacdCurrent<0 and MacdCurrent>SignalCurrent and
      MacdPrevious<SignalPrevious and Abs(MacdCurrent)>(MACDOpenLevel*Point) and
      MaCurrent>MaPrevious then
     {
      SetOrder(OP_BUY,Lots,Ask,3,0,Ask+TakeProfit*Point,RED); // executing
      Exit; // exiting, since after the execution of a trade
            // there is a 10-second trading timeout
     };

   // checking for the possibility of taking a short position (SELL)
   If MacdCurrent>0 and MacdCurrent<SignalCurrent and
      MacdPrevious>SignalPrevious and MacdCurrent>(MACDOpenLevel*Point) and
      MaCurrent<MaPrevious then
     {
      SetOrder(OP_SELL,Lots,Bid,3,0,Bid-TakeProfit*Point,RED); // executing
      Exit; // exiting
         };
   // here we completed the check for the possibility of opening new positions.
   // no new positions were opened and we simply exit the programme using the Exit command, a
   // there is nothing to analyze
```

```
    Exit;
   };

// we come over to an important part of the Expert Advisor - the control of open positions
// 'it is important to enter the market correctly, but it is even more important to exit it..
for cnt=1 to TotalTrades
   {
    if OrderValue(cnt,VAL_TYPE)<=OP_SELL and   // is this an open position? OP_BUY or OP_SELL
       OrderValue(cnt,VAL_SYMBOL)=Symbol then  // does the instrument match?
      {
       If OrderValue(cnt,VAL_TYPE)=OP_BUY then // long position opened
         {
          // we check - maybe, it's already time to close it?
          If MacdCurrent>0 and MacdCurrent<SignalCurrent and
             MacdPrevious>SignalPrevious and MacdCurrent>(MACDCloseLevel*Point) then
            {
             CloseOrder(OrderValue(cnt,VAL_TICKET),OrderValue(cnt,VAL_LOTS),Bid,3,Violet);
             Exit; // exiting
            };

          // we check - maybe, we already may or it's already time to set a trailing stop?
          If TrailingStop>0 then  // the user has put a trailing stop in his settings
            {                     // so, we set out to check it
             If(Bid-OrderValue(cnt,VAL_OPENPRICE))>(Point*TrailingStop) then
               {
                If OrderValue(cnt,VAL_STOPLOSS)<(Bid-Point*TrailingStop) then
                  {
                   ModifyOrder(OrderValue(cnt,VAL_TICKET),OrderValue(cnt,VAL_OPENPRICE),
                               Bid-Point*TrailingStop,OrderValue(cnt,VAL_TAKEPROFIT),Red);
                   Exit;
                  };
               };
            };
         }
       else // otherwise it is a short position
         {
          // we check - maybe, it's already time to close it?
          If MacdCurrent<0 and MacdCurrent>SignalCurrent and
             MacdPrevious<SignalPrevious and Abs(MacdCurrent)>(MACDCloseLevel*Point) then
            {
             CloseOrder(OrderValue(cnt,VAL_TICKET),OrderValue(cnt,VAL_LOTS),Ask,3,Violet);
             Exit; // exiting
            };

          // we check - maybe, we already may or it's already time to set a trailing stop?
          If TrailingStop>0 then  // the user has put a trailing stop in his settings
            {
             // so, we set out to check it
             If(OrderValue(cnt,VAL_OPENPRICE)-Ask)>(Point*TrailingStop) then
               {
                If OrderValue(cnt,VAL_STOPLOSS)=0 or
                   OrderValue(cnt,VAL_STOPLOSS)>(Ask+Point*TrailingStop) then
                  {
                   ModifyOrder(OrderValue(cnt,VAL_TICKET),OrderValue(cnt,VAL_OPENPRICE),
                               Ask+Point*TrailingStop,OrderValue(cnt,VAL_TAKEPROFIT),Red);
                   Exit;
                  };
               };
            };
         };
      };
   };
// the end.
```

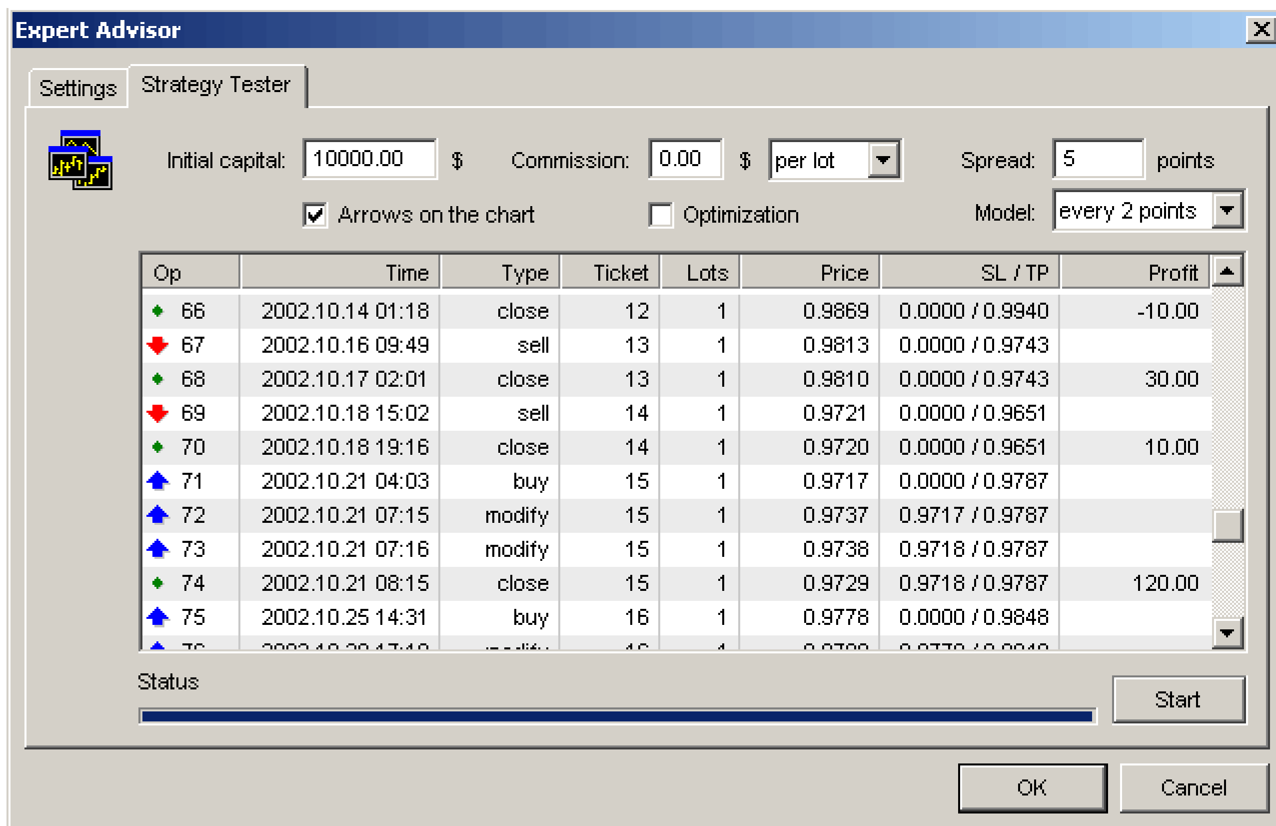
To complete setting up the Expert Advisor we need only to specify the values of the external variables **Lots = 1**, **Stop Loss (S/L) = 0** (not used), **Take Profit (T/P) = 120** (appropriate for one-hour intervals), **Trailing Stop (T/S)**

= 30. Of course, you can set your own values. Press the Verify button and, if there isn't any error message (by the way, you can copy the text from the above programme printout against the grey background right into the MetaEditor), press the Save button to save the Expert Advisor.

Step IV: Testing the Expert Advisor on historical data



We have written the Expert Advisor - now we are impatient to evaluate it by testing it on historical data. Let's take as an example 15-minute intervals for EUR/USD, approximately 4000 bars.
Let's open EURUSD, M15 chart in MetaTrader, attach the MACD Sample Expert Advisor to the chart using the Attach to a chart command (by selecting the MACD Sample line by the mouse pointer in the Navigator window, pressing the right mouse button and selecting the command in the appearing menu). After that we go the the Expert Advisor settings, where we can change the predefined external variables Lots, Stop loss, Take profit, Trailing stop, as well as the user-defined variables. To allow the Expert Advisor not only to advise, but also to gamble on the trading account in real-time on its own, you need to activate the Allow Live trading button. We are going to do testing on historical data, though, so we leave the settings unchanged, switch to the Strategy Tester tab, activate the Arrows on the chart tick (to be able to see the arrows on the chart) and start our test with the Start button:



Features of Expert Advisors

Writing and testing experts in MetaTrader trading system has a number of features.

- Before opening a position it is necessary to check if there is any free money at the account. If there is not enough money at the account the operation of opening a position will end unsuccessfully. And at the same time, when being tested the FreeMargin value must necessarily be above 1000 because when being tested one lot price makes 1000.

```
If FreeMargin < 1000 Then Exit; // no money - exit
```

- After having opened, closed or changed a position or after having deleted a pending order (i.e., after having used any of the following commands: SetOrder, CloseOrder, ModifyOrder or DeleteOrder), it is recommended to finish the expert program using the Exit instruction, as after completing the operation a 10-seconds-timeout begins which is used for processing trading operations. But for all that, this 10-seconds-timeout does not function during the testing (it is possible to process several operations one after another), and, if one does not finish the expert running by executing the Exit instruction after having completed the operation, the testing results can differ from those of the real expert trading.

```
SetOrder(OP_SELL,Lots,Bid,3,0,Bid-TakeProfit*Point,RED); // execute
Exit; // exit
```

  To prevent a possibility of processing several trade operations with an interval of less than 10 seconds during the testing it is also possible to check before processing the next operation that no less than 10 seconds have passed after completing the last trade operation.

```
// check that the current time is more than the time taken for completing the last trade operation more
than 10 seconds
If CurTime > LastTradeTime + 10 Then Begin
   SetOrder(OP_SELL,Lots,Bid,3,0,Bid-TakeProfit*Point,RED); // execute
   Exit;
End;
```

- The admittance for historical data can be obtained using indexing predefined variables Open, Close, High, Low, Volume. Index in this case is the number of periods necessary to be counted back.

```
// if Close at the last bar is less than Close at the bar before last
If Close[1] < Close[2] Then Exit;
```

- Experts testing in MetaTrader trading system supports 4 models.
  - **OHLC points** (Open/High/Low/Close). When testing an expert only Open, High, Low, Close prices are used. This model of testing is the quickest one. But the results of the testing can differ from those of the real expert trading.
  - **Every spread/2 points**. When testing an expert the candle development having the "spread/2"(usually 2) points step is simulated.
  - **Every 1 point**. When testing an expert the candle development having the 1 point step is simulated. This model of testing is the slowest one. But the results of the testing are identical with those of the real expert trading (when keeping the 10 seconds interval between completing trading operations).

- When writing and testing an expert, like testing any other program, it is sometimes necessary to display some additional debugging information. The MQL II language gives a number of opportunities to display such information.

    - The Alert function displays a dialogue box containing data defined by the user.
      ```
      Alert("Free margin is ", FreeMargin);
      ```
    - The Comment function shows the data defined by the user in the upper left corner of the chart. The "\n" char sequence is used for line feed.
      ```
      Comment("Free margin is ", FreeMargin, "\nEquity is ", Equity);
      ```
    - The Print function prints data defined by the user in the log file.
      ```
      Print("Total trades are ", TotalTrades, "; Equity is ", Equity, "; Credit is ", Credit);
      ```
    - The PrintTrade function prints data of selected trade position in the log file.
      ```
      PrintTrade(1);
      ```

- When testing an expert the test results are stored in files having ".log" extension in a subdirectory of the "logs"-directory where you have already installed your MetaTrader trading system. When you often test experts do not forget to delete log files periodically as they can have a size of several megabytes.

- It occurs that it is necessary to perform a large preparatory work before the experts start working. This work is so large that the expert will close forcedly with the error of loop detected. (It is necessary to note that the expert runs with each price tick and may not work long). But you can perform the preparatory work, for example, initialization of any array, in several movements. For example, in this way:
  ```
  var: cnt(0), initvalue(0);
  array: arr[6000](0);

  SetLoopCount( 50000 ); // for example only. illustrates an initialization in 3 times

  If initvalue <= 6000 Then Begin
      print( "beginning from ", initvalue );
      For cnt = initvalue To 6000 Begin // initialization loop
          arr[cnt] = cnt;
          // strongly recommended last instruction in the initialization loop as follow
          initvalue = cnt + 1;
      End;
      print( "stopped with ", initvalue, " -- initialization is complete." );
  End
  Else print( "initialization is complete" );
  ```
  If it is necessary to initialize several arrays you should use several variables. For example, initvalue, initvalue2, initvalue3, etc.

- How to assess the beginning of the next bar? (This is necessary to know that the previous bar has just been formed.) There are several ways:
  ```
  Variable: prevbars(0);
  ...
  If prevbars = Bars Then exit;
  prevbars = Bars;
  ...
  ```
  This method can sometimes fail when spooling the history. I.e., the number of bars has changed, and the "previous" bar has not been formed yet. In this case, it is possible to complicate checking the difference between values being equal to 1.

  The next way is based on the fact that the value of Volume is formed on the base of the number of ticks having come for each bar, and the first tick means that the value of the bar being newly formed the Volume value is equal to 1:

```
If Volume > 1 Then exit;
...
```

This method can fail when price ticks come too intensively. As a matter of fact, the processing of the price ticks coming is performed in a separate traffic. And if this traffic is occupied during the next tick coming, this coming tick will not be processed to prevent overloading the computing power! In this case, it is possible to complicate the checking using storage of the previous Volume value.

The third way is based on the bar opening time:

```
Variable: prevtime(0);
...
If prevtime = Time Then exit;
prevtime = Time;
...
```

This method is the most effective. It will work under any conditions.

- An example of working with the file is given below:

```
vars : handle(0), cnt1(0), cnt2(0), position(0), size(0);
vars : string(""), number(0);

handle = FileOpen( "test", "," );
print( "file size is ", FileSize( handle ) );
size = 0;
while not IsFileEnded( handle ) begin
   string = FileReadString( handle );
   number = FileReadNumber( handle );
   // lines counting
   if IsFileLineEnded( handle ) then size = size + 1;
end;
print( "file has ", size, " lines" );
if size < 20 then begin
   cnt2 = 0;
   for cnt1 = size to 19 begin
      FileWrite(handle, "teststring"+cnt1, cnt1+1);
      cnt2 = cnt2 + 1;
      size = size + 1;
      if cnt2 = 5 then break;
   end;
   print( "there are ", cnt2, " lines added" );
 print( "file size is ", FileSize( handle ) );
end;

position = FileSeek( handle, 0, SEEK_SET );
// file reading from begin
for cnt1=1 to size begin
   print( FileReadString( handle ), ";", FileReadNumber( handle ) );
   If IsFileEnded( handle ) then break;
end;
FileClose(handle);
```

Here are several explanations to the code. The file will be opened first, then the line-by-line reading should be performed. It is presumed that the structure of each line of the file consists of one text variable and one numeric

variable. If the number of lines being counted is less than 20 the next portion of 5 lines will be added. Since the pointer of the current position of reading-recording is at the end of the file after the lines have been counted, the recording of the next record will be made to the end of the file, after that the pointer of the current position will be moved to the end of the file. If the record should be done to the end of the file immediately after it has been opened it is necessary to use the following function:

```
position = FileSeek( handle, 0, SEEK_END );
```

after that the data should be stored in the file.

In the second part of the code the reading of records from the file will be conducted from its very beginning.

Custom Indicators and Functions

Writing user indicators and functions in MetaTrader trading system has a number of features, as well:

- Parameters to be taken into consideration for user indicators and functions should be described in the Inputs section and be of numeric data type.
```
Inputs: nPeriod(13),nDeviation(0.1),nAccountedBars(300);
```

- Parameters will be sent to the function of custom indicator calculation in the order in which they are described. For example, call the custom indicator having the parameters listed above will appear as follows:
```
iCustom( "SomeIndicator", 13, 0.1, 300, MODE_FIRST, 0 );
```

- Strictly speaking, parameters should not be necessarily sent to the function. If the function does not have an Inputs section it is no use to send the parameters. Or one can use the initial values being applied for describing parameters. For example, call the same custom indicator without parameters will appear as follows:
```
iCustom( "SomeIndicator", MODE_FIRST, 0 );
```
This means that those values will be used which are stored in variables described in the Inputs section,i.e. 13, 0.1 and 300.

- What is the difference between the function of custom indicator calculation and a simple user function? It is defined by the methods of call. The function of custom indicator is called once when recalculation of the whole array of price data. The simple user function is called during the expert program working, i.e. with each tick. The function of custom indicator calculation has access to two arrays storing the results of calculation with the help of functions named SetIndexValue, SetIndexValue2, GetIndexValue, GetIndexValue2. And the simple user function has access to these arrays with the help of iCustom function. The function of custom indicator calculation can not call trading functions and functions of call of user functions. User function should be finished with the call of the Return function in order to send the value calculated to the expert-caller.

- There is a special value of index array which means the absence of data, it is 0. If zero value is significant in any of the custom indicators, it is necessary to use a very small value, for example, 0.00000001. It is necessary to note that when displaying the custom indicators in the pop-up prompts, the accuracy of 4 signs after the dot in decimal fraction will be used. Indicator placed in the price chart can not have negative data as they can not be displayed in any case. That is why negative values of the indicator placed in the price chart will also be considered as special ones. (The features of use of special values of custom indicators being placed in the price chart are discussed below.)

- The values of the variables in the function of custom indicator calculation, and the data arrays with the results of

custom indicator calculation will be initialized at the first run of the indicator, and at the period or instrument change at the chart. It means, all values will be stored between initializations! When custom indicator is programmed in a proper and accurate way it is possible to provide the possibility of calculation of only last, not calculated values, data arrays of the indicator. An example of an indicator of a simple moving average:

```
/*[[
    Name := SimpleMA
    Author := Copyright © 2003, MetaQuotes Software Corp.
    Link := http://www.metaquotes.ru/
    Separate Window := No
    First Color := Red
    First Draw Type := Line
    Use Second Data := No
]]*/
Inputs : MAPeriod(13);
Variables : shift(0), cnt(0), loopbegin(0), first(True), prevbars(0), sum(0);

SetLoopCount(0);
// initial checkings
If MAPeriod < 1 Then Exit;
// check for additional bars loading or total reloading
If Bars < prevbars Or Bars-prevbars>1 Then first = True;
prevbars = Bars;
If first Then Begin
    // loopbegin prevent couning of counted bars exclude current
    loopbegin = Bars-MAPeriod-1;
    If loopbegin < 0 Then Exit; // not enough bars for counting
    first = False;
End;

// moving average
loopbegin = loopbegin+1; // current bar is to be recounted too
For shift = loopbegin Downto 0 Begin
    sum = 0;
    For cnt = 0 To MAPeriod-1 Begin
        sum = sum + Close[shift+cnt];
    End;
    SetIndexValue(shift,sum/MAPeriod);
    loopbegin = loopbegin-1; // prevent to previous bars recounting
End;
```

Some explanations for the code. In the first variable after initialization the True value is stored. As soon as the indicator runs for the first time the check of the first variable will be conducted and the value of the loopbegin variable will be established on the assumption that the cycle would run from the data beginning. When the cycle is running the loopbegin variable is modified and contents in fact the value of the index of the last value calculated in the indicator array. In the case of "loop detected", the indicator expert will stop its work. But the first variable is established in False, it means that at the next start the loopbegin variable will not be established at the beginning of the indicator array. It means that the new calculation will be continued from the point where the previous calculation has been made. At the end, all the bars will be calculated. But if we need to recalculate the indicator value at the current bar with the coming of each new quote we modify the loopbegin variable before the cycle starts, increasing it by 1 to realize at least 1 cycle iteration.

If the historical data were spooled (Refresh Chart) or reloaded (Erase and Refresh Chart) the condition of the

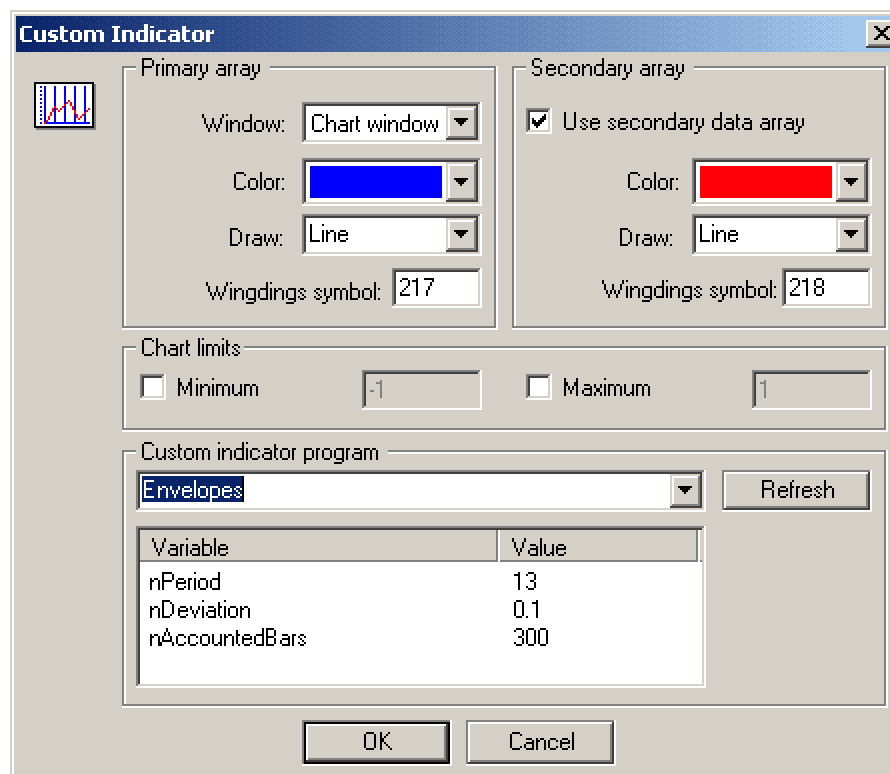
comparing of the current number of bars with the previous number of those would allow us to recalculate indicator on all data.

- Custom indicator function can be called not only from the expert program, but also from the Navigator window when selecting Custom Indicator. In this connection, there are several features of designing such a program. An example of the Envelopes indicator:

```
/*[[
   Name := Envelopes
   Author := Copyright © 2003, MetaQuotes Software Corp.
   Link := http://www.metaquotes.net/
   Notes := Sample Custom Indicator program
   Separate Window := No
   First Color := Blue
   First Draw Type := Line
   Use Second Data := Yes
   Second Color := Red
   Second Draw Type := Line
]]*/
Inputs : nPeriod(13),nDeviation(0.1),nAccountedBars(300);
Vars : CurrentBar(0),shift(0),i(0),sum(0),UpperBand(0),LowerBand(0),BeginBar(0);
Vars : prevBars(0);

If prevBars = Bars Then Exit;
prevBars = Bars;
SetLoopCount(0);

BeginBar = Bars - nAccountedBars;
If BeginBar < 0 then BeginBar = 0;
For CurrentBar = 0 To BeginBar-1 Begin
   shift = Bars-1-CurrentBar;
   SetIndexValue(shift, 0);
   SetIndexValue2(shift, 0);
End;

For CurrentBar = BeginBar To Bars-1 Begin
   shift = Bars - 1 - CurrentBar;
   If CurrentBar < nPeriod-1 Then Begin;
      SetIndexValue(shift, 0);
      SetIndexValue2(shift, 0);
      Continue;
   End;

   sum = 0;
   For i=0 To nPeriod-1 Begin
      sum = sum + Close[shift+i];
   End;
   sum = sum / nPeriod;
   UpperBand = sum * (1 + nDeviation/100);
   LowerBand = sum * (1 - nDeviation/100);
   SetIndexValue(shift, UpperBand);
```

```
        SetIndexValue2(shift, LowerBand);
    End;
```

As you can see the Experts Wizard creates the description of custom indicator differing from the description of the expert program.
Separate Window := No - means that the indicator chart will be developed in the same window where the price chart is developed. If the values calculated with the custom indicator differ from the current prices very much, the indicator chart must be shown in a separate window.
Draw Type := Line - means that the indicator chart will be developed as a line. It is also possible to develop a histogram (Histogram) or to show a separate symbol (Symbol; in this case, the symbol will be set as a number defining the position of the symbol in the Wingdings table. For example, First Symbol := 217).
UUse Second Data := Yes - means that the indicator calculation results in 2 charts, and not the only one (another example: MACD histogram and signal line). First Color, First Draw Type and First Symbol belong to the first indicator chart, the Second Color, Second Draw Type и Second Symbol belong to the second one.
If the indicator is shown in a separate window and the limiting values of the minimum and/or maximum are known beforehand, these limiting values can be defined to accelerate the displaying and, in some case, to make the displaying more beautiful (this especially concerns histograms). Experts Master will, for example, design it, as follows:
Minimum Chart Limits := -1.000000
Maximum Chart Limits := 1.000000
All parameters listed and made with the Expert Master help in quick setting of Custom Indicator.



- As it was mentioned above, the user expert chart can be displayed as a histogram. There is a feature of displaying the histogram in the price chart window. Both data arrays of the Custom indicator participate in the process of histogram displaying. The line will be drawn at the chart from the first value to the second corresponding one. If the first value is more than the second one the line will be drawn with the first color. Otherwise, it will be drawn with the second color. This feature can be used when coloring the bars with different

colors.

```
/*[[
   Name := ColoredBars
   Author := Copyright © 2003, MetaQuotes Software Corp.
   Link := http://www.metaquotes.net/
   Notes := Sample Custom Indicator program
   Separate Window := No
   First Color := Blue
   First Draw Type := Histogram
   Use Second Data := Yes
   Second Color := Pink
   Second Draw Type := Histogram
]]*/
Variables : CurrentBar(0),shift(0),nBWMFI(0),prevBWMFI(0),value1(0),value2(0);

SetLoopCount(0);
For CurrentBar = 0 To Bars-1 Begin
   shift = Bars-1-CurrentBar;
   nBWMFI = (High[shift] - Low[shift]) / Volume[shift];
   If CurrentBar < 1 Then Begin
      SetIndexValue(shift, 0);
      SetIndexValue2(shift, 0);
      prevBWMFI = nBWMFI;
      Continue;
   End;

   value1 = 0;
   value2 = 0;
   If nBWMFI > prevBWMFI and Volume[shift] < Volume[shift+1] Then Begin
      value1 = High[shift];
      value2 = Low[shift];
   End;
   If nBWMFI < prevBWMFI and Volume[shift] > Volume[shift+1] Then Begin
      value1 = Low[shift];
      value2 = High[shift];
   End;
   SetIndexValue(shift, value1);
   SetIndexValue2(shift, value2);
   prevBWMFI = nBWMFI;
End;
```

The example given above illustrates coloring the bars in Bill Williams style. If necessary, it is possible to prepare another program of custom indicator to color bars with two other colors corresponding with other conditions.

- The features of special values processing were mentioned above which had been less or equal to 0 and concerned the "line" custom indicators placed in the price chart. 0 means the absence of data. The line will be plotted between points having positive values. A negative value means the end of the previous line. I.e., lines can be interrupted. As an illustration of the "absence of data" an example of the ZigZag indicator is given below which plots lines between local price extremes.

```
/*[[
```

```
    Name := ZigZag
    Author := Copyright © 2003, MetaQuotes Software Corp.
    Link := http://www.metaquotes.ru
    Separate Window := No
    First Color := Blue
    First Draw Type := Line
    Use Second Data := No
]]*/
Inputs: depth(12),deviation(5),backstep(3);
Variables : shift(0),lasthigh(-1),lastlow(-1),lasthighpos(0),lastlowpos(0);
Variables : val(0),back(0),res(0);
Variables : curlow(0),curhigh(0);

SetLoopCount(0);
lasthigh=-1;
lastlow=-1;

for shift = Bars-300 downto 0
{
    //--- low
    val=Low[Lowest(MODE_LOW,shift+depth-1,depth)];
    if val==lastlow then val=0
    else
     {
        lastlow=val;
        if (Low[shift]-val)>(deviation*Point) then val=0
        else
         {
            for back=1 to backstep
             {
                res=GetIndexValue(shift+back);
                if res!=0 and res>val then SetIndexValue(shift+back,0);
             };
         };
     };
    SetIndexValue(shift,val);
    //--- high
    val=High[Highest(MODE_HIGH,shift+depth-1,depth)];
    if val==lasthigh then val=0
    else
     {
        lasthigh=val;
        if (val-High[shift])>(deviation*Point) then val=0
        else
         {
            for back=1 to backstep
             {
                res=GetIndexValue2(shift+back);
                if res!=0 and res<val then SetIndexValue2(shift+back,0);
             };
```

```
      };
    };
   SetIndexValue2(shift,val);
};

// final cutting
lasthigh=-1; lasthighpos=-1;
lastlow=-1; lastlowpos=-1;

for shift = Bars-300 downto 0
{
   curlow=GetIndexValue(shift);
   curhigh=GetIndexValue2(shift);
   if curlow==0 & curhigh==0 then continue;
   //---
   if curhigh!=0 then
    {
      if lasthigh>0 then
       {
         if lasthigh<curhigh then SetIndexValue2(lasthighpos,0)
         else SetIndexValue2(shift,0);
       };
      //---
      if lasthigh<curhigh then
       {
         lasthigh=curhigh;
         lasthighpos=shift;
       };
      lastlow=-1;
    };
   if curlow!=0 then
    {
      if lastlow>0 then
       {
         if lastlow>curlow then SetIndexValue(lastlowpos,0)
         else SetIndexValue(shift,0);
       };
      //---
      if curlow<lastlow | lastlow<0 then
       {
         lastlow=curlow;
         lastlowpos=shift;
       };
      lasthigh=-1;
    };
};

for shift = Bars-300 downto 0
{
   res=GetIndexValue2(shift);
```

```
    if res!=0 then SetIndexValue(shift,res);
  };
```

As a result, the indicator given above fills the first indicator data array with the extreme values. The values between extremes are zero values. The lines are drawn between extremes (i.e., between positive values).

- As an illustration of symbols displaying in defining the position of the price chart a program of the simplest calculation of Bill Williams fractals is given below:

```
/*[[
    Name := Fractals
    Author := Copyright © 2003, MetaQuotes Software Corp.
    Link := http://www.metaquotes.net/
    Notes := Sample Custom Indicator program
    Separate Window := No
    First Color := Blue
    First Draw Type := Symbol
    First Symbol := 217
    Use Second Data := Yes
    Second Color := Red
    Second Draw Type := Symbol
    Second Symbol := 218
]]*/
Variable : value(0),CurrentBar(0), price(0);

SetLoopCount(0);
For CurrentBar = 0 To Bars-1 Begin
    If CurrentBar < 2 Or CurrentBar >= Bars-2 Then Begin
        SetIndexValue(CurrentBar, 0);
        SetIndexValue2(CurrentBar, 0);
        Continue;
    End;

    value = 0;
    price = High[CurrentBar];
    If price>High[CurrentBar+1] and price>High[CurrentBar+2] and price>High[CurrentBar-1] and
price>High[CurrentBar-2] then
        value = price;
    SetIndexValue(CurrentBar, value);

    value = 0;
    price = Low[CurrentBar];
    If price<Low[CurrentBar+1] and price<Low[CurrentBar+2] and price<Low[CurrentBar-1] and
price<Low[CurrentBar-2] then
        value = price;
    SetIndexValue2(CurrentBar, value);
End;
```

- A user function example. Before using the function, it is necessary to be sure that it has been successfully translated, and that there is a corresponding exp-file in the expert catalogue.

```
/*[[
    Name := UserFunc
```

```
        Author := Copyright © 2003, Metaquotes Software Corp.
        Link := http://www.metaquotes.ru/
    ]]*/
    Input : parameter(0);

    print( "Input parameter is ", parameter );
    return( parameter );
```

User Function call sample:

```
    Variable: ReturnValue(0);

    ReturnValue = UserFunction( "UserFunc", Close );
    print( "Return value is ", ReturnValue );
```

- It must be noted that the **overflow of custom indicators, improperly written user functions can slow down the functioning of the client terminal!**

To Beginning