MetaTrader 3.85: Expert Advisor User's Guide

The Expert Advisor is a revolutionary trading tool that allows clients to program their own trading strategies.

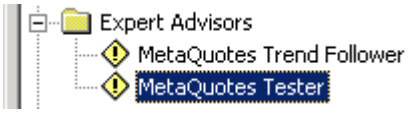
Expert Advisors are used to automate the trading process and relieve the trader of the routine functions of continuous market monitoring. Many professional traders employ multiple trading systems enabling them to operate in diverse markets and under a variety of environments. Usually they write and test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

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 an automated trading system (ATS) 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 execute trades on 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 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 if 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 program.

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.

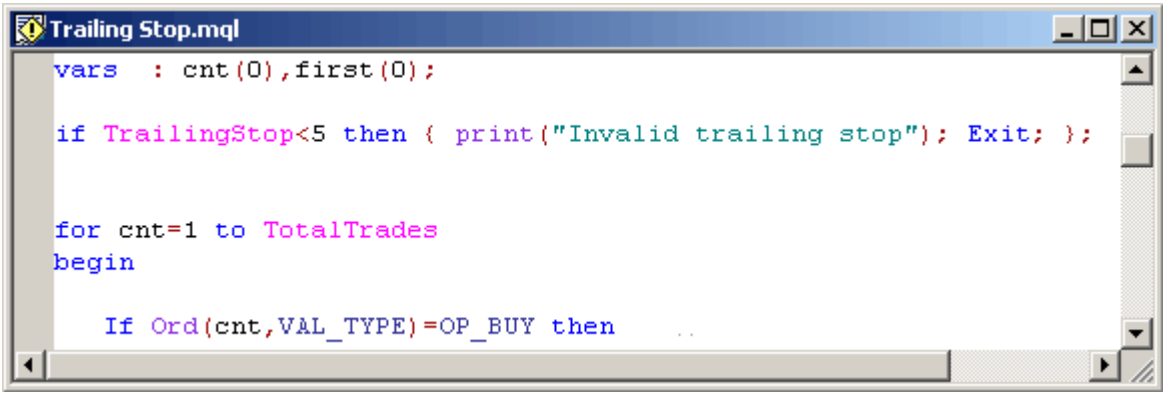


MetaTrader stores Expert Advisors as *.MQL(source text) and *.EXP (executable code) files in /Experts subdirectory of the root directory of the program. A trader is able to maintain an unlimited number of Expert Advisors which can be easily managed through the Navigator window.

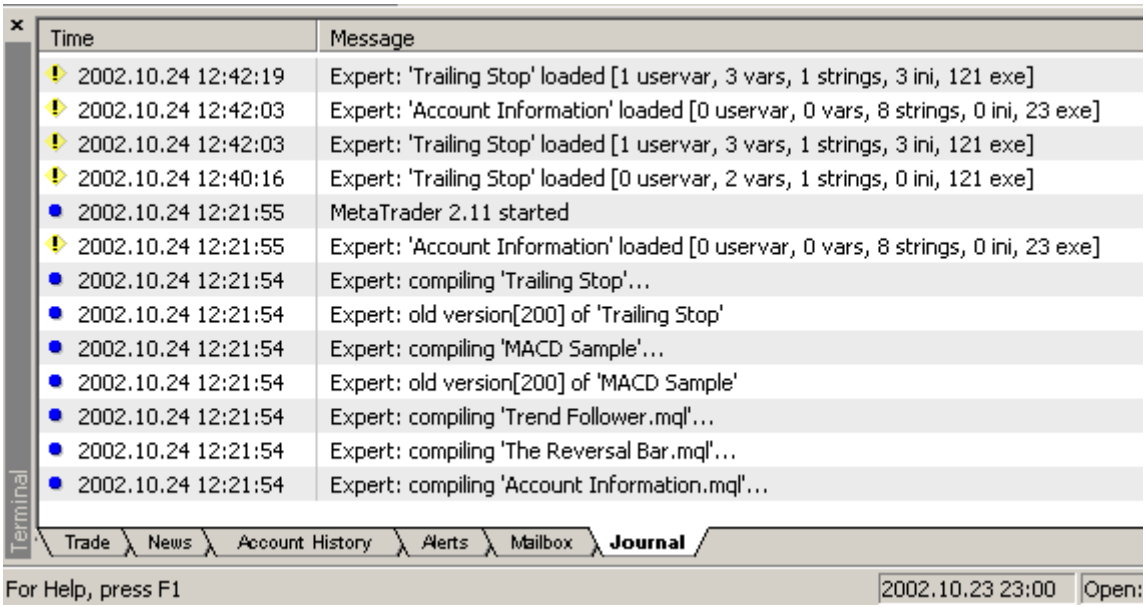
Procedure of creating the custom Expert Advisors and linking them to the trading terminal is described in detail in the MetaTrader User Guide

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.

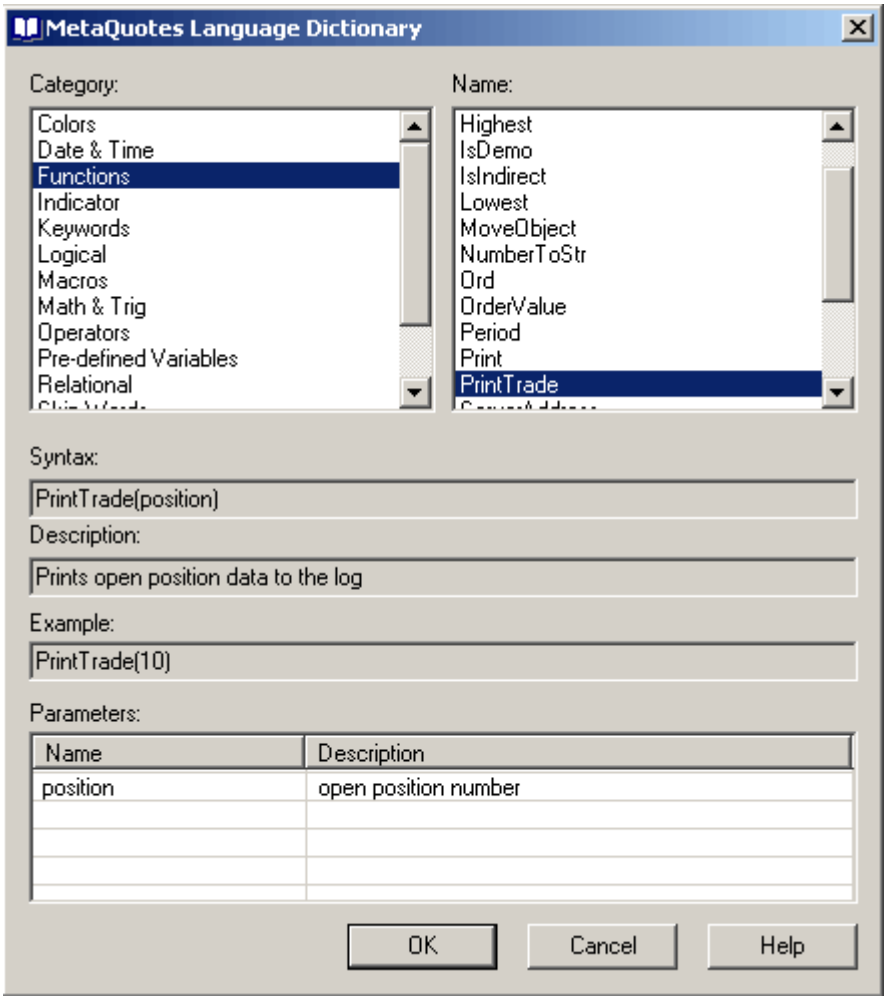
The program code is written using the MetaEditor advisor text editor, which is capable of highlighting different structures of MQL II language in different colors, thus helping the user through the text of the expert system. Comments start with the // symbol (double slash). Comments can also be marked 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 catalog. 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 program.

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		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, Ceil, CloseOrder, Comment, Cos, CurTime, Day,

DayOfWeek, DeleteOrder, Exp, Floor, Highest, Hour, iADX, iATR, iBANDS, iCCI, iMA, iMACD, iMFI, iMOM, iRSI, iSAR, iSTO, iWPR, IsDemo, IsIndirect, LastTradeTime, Log, Lowest, Minute, Mod, ModifyOrder, Month, MoveObject, NumberToStr, Ord, OrderValue, Period, Pow, Print, PrintTrade, Rand, Round, ServerAddress, SetArrow, SetObjectText, SetOrder, SetText, Sin, Sqrt, Srand, Symbol, Tan, TimeToStr, 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_EMA, MODE_HIGH, MODE_LOW, MODE_MAIN, MODE_MINUSDI, MODE_OPEN, MODE_PLUSDI, MODE_SIGNAL, MODE_SMA, MODE_STOPLOSS, MODE_TAKEPROFIT, MODE_VOLUME, MODE_WMA, OBJ_HLINE, OBJ_SYMBOL, OBJ_TEXT, OBJ_TRENDLINE, OBJ_VLINE, OP_BUY, OP_BUYLIMIT, OP_BUYSTOP, OP_SELL, OP_SELLLIMIT, OP_SELLSTOP, SYMBOL_ARROWDOWN, SYMBOL_ARROWUP, SYMBOL_CHECKSIGN, SYMBOL_STOPSIGN, SYMBOL_THUMBSDOWN, SYMBOL_THUMBSUP, VAL_CLOSEPRICE, VAL_CLOSETIME, VAL_COMISSION, VAL_COMMENT, 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

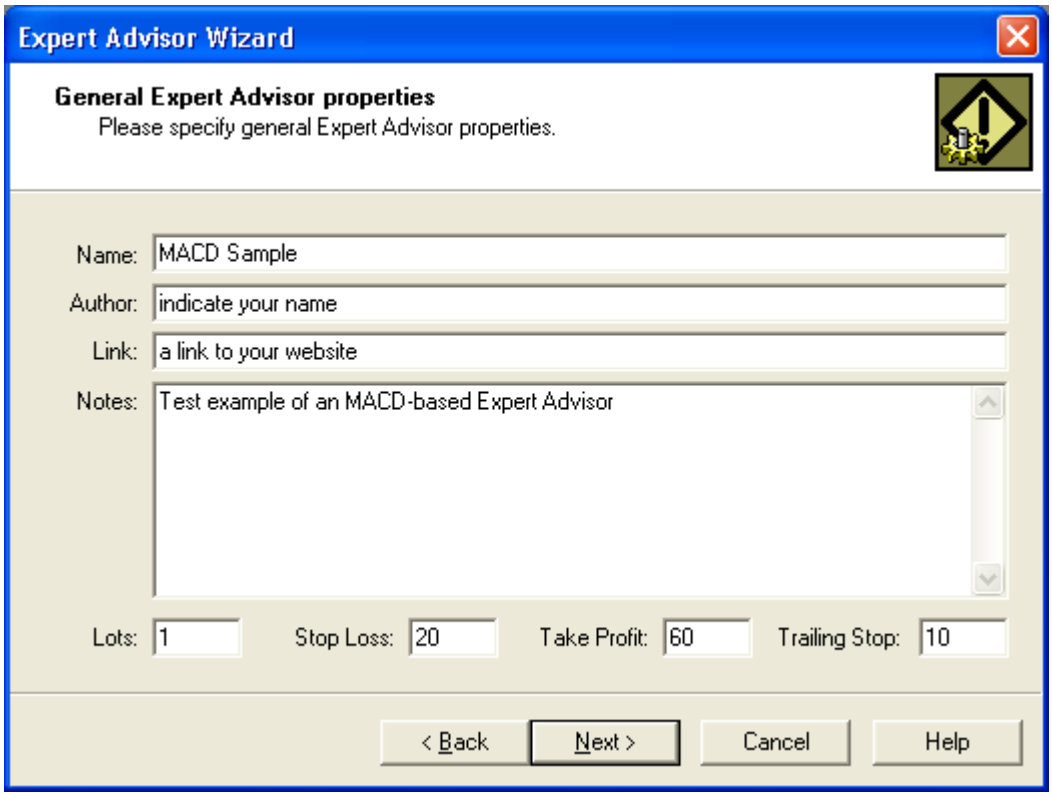
7. Colors.
AliceBlue, AntiqueWhite, Aqua, Aquamarine, Azure, Beige, Bisque, Black, BlanchedAlmond, Blue, BlueViolet, Brown, BurlyWood, CadetBlue, Chartreuse, Chocolate, Coral, CornflowerBlue, Cornsilk, Crimson, Cyan, DarkBlue, DarkCyan, DarkGoldenrod, DarkGray, DarkGreen, DarkKhaki, DarkMagenta, DarkOliveGreen, 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.

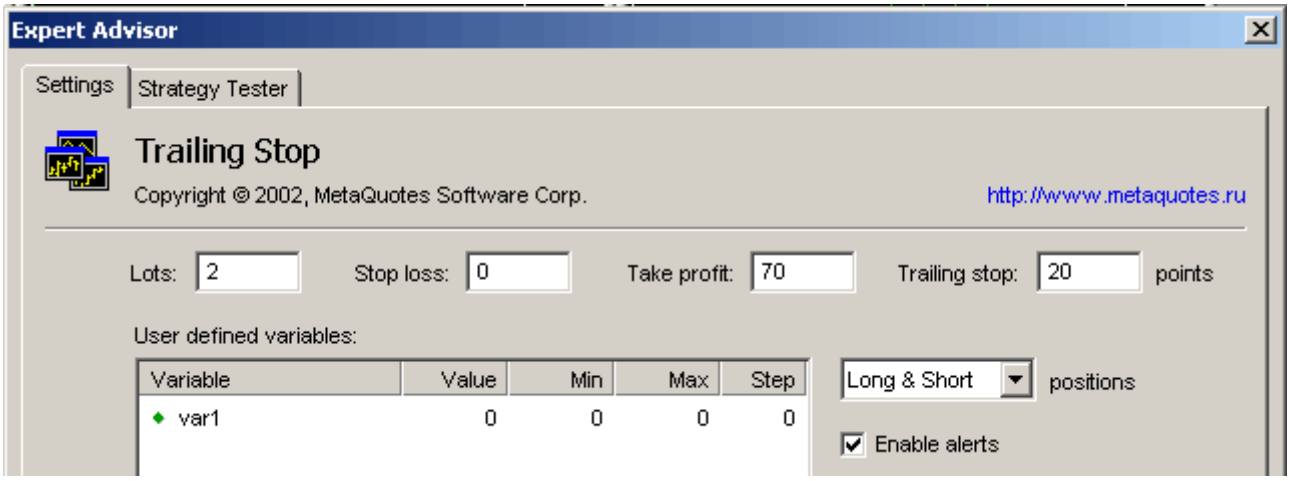
All the reserved words are case-insensitive, i.e. they may be written in lower as well as in upper case.

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.



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 program.



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 program using reserved words Variable, Array and Define. It is allowed to use these words in plural, i.e. Variables, Arrays and Defines. 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 program.

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 program. 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.

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 program.

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 program. 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 Statement1
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 (the Buyer's price);
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 - Bid Price (the Seller's price);
Close - Closing 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 - Opening 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;
PriceBid - current Bid Price as shown in the "Market Watch" window;
PriceHigh - maximum Ask Price over the current 24-hour period;
PriceLow - minimum Ask Price over the current 24-hour period;
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, trading functions, time functions, mathematical and trigonometric functions, data conversion and output functions, etc.

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

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

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

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

CloseOrder - position closing.
Syntax: CloseOrder(order, lots, price, slippage, color)
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.
Syntax: Comment(...)
Any non-zero number of parameters is possible.

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

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

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

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

Highest - returns the maximum value of Open, Low, High, Close or Volume (depending on the "type" parameter) over the specified number of periods.

Syntax: Highest(type, beginbar, periods)
Parameters:
type - a returned variable, may take one of the following values: MODE_OPEN, MODE_LOW, MODE_HIGH, MODE_CLOSE, MODE_VOLUMEbeginbar - a shift showing the bar, relative to the current bar, that the data should be taken from.
periods - number of periods on which the calculation is carried out.

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

iADX - index of the average directed motion.
Syntax: iADX(period, mode, shift)
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.

iATR - indicator of the average true interval.
Syntax: iATR(period, shift)
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 - Bollinger band indicator.
Syntax: iBANDS(period, deviation, mode, shift)
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.

iCCI - trading channel index.
Syntax: iCCI(period, shift)
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.

iMA - moving average indicator.
Syntax: iMA(period, mode, shift)
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.

iMACD - moving averages convergence/divergence indicator.
Syntax: iMACD(fast_ema_period, slow_ema_period, signal_period, mode, shift)
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.

iMFI - cashflow index.
Syntax: iMFI(period, shift)
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 - Momentum indicator.
Syntax: iMOM(period, shift)
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.

iRSI - relative strenghth index.
Syntax: iRSI(period, shift)
Parameters:
period - number of periods for calculation;

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

iSTO - "Stochastic oscillator" indicator .
Syntax: iSTO(%Kperiod, %Dperiod, slowing, method, mode, shift)
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 - Williams percentage range indicator.

Syntax: iWPR(period, shift)
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 training account. Otherwise returns False value.

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

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

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

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

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

ModifyOrder - modification of characteristics for the previously opened position or a delayed order.
Syntax: ModifyOrder(order, price, stoploss, takeprofit, color)
Parameters:
order - number or order of the opened or delayed position;
price - new price (only for a delayed position!);
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 - moving or creation of a named object.
Syntax: MoveObject(name, type, time, price, time2, price2)
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);
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

NumberToStr - 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.
Syntax: NumberToStr(number, precision)
Parameters:
number - specified numerical value;
precision - precision format, number of digits after the decimal point.

OrderValue - returns one of the specified values of the order.
Syntax: OrderValue(position, mode)
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 - returns the number produced by raising the nBaseExpression number to the nExponentExpression power, given as parameters.
Syntax: Pow(nBaseExpression, nExponentExpression)
Parameters: numerical value1, numerical value2.

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

PrintTrade - stores the details of a particular position in the log.
Syntax: PrintTrade(position)
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 program.

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

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

SetArrow - setting the pictographic symbol on the chart.
Syntax: SetArrow(time, price, symbol, color)
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.

SetObjectText - assign a text string to a specified object.
Syntax: SetObjectText(name, text, font, size, color)
Parameters:
name - object name;
text - specified text;
font - font name;
size - font size;
color - text color.

SetOrder - main function used to open a position or set a delayed order
Syntax: SetOrder(operation, lots, price, slippage, stoploss, takeprofit, color)
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 - put a text string on the chart in the specified position.
Syntax: SetText(time, price, string, color)
Parameters:
time - chart reference point by time;
price - chart reference point by price;
string - text string;
color - text color.

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

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

Srand - 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 program. 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.
Syntax: Srand(Time)
Parameter: positive numerical value.

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

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

TimeToStr - 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..
Syntax: TimeToStr(Time)
Parameter: positive numerical value.

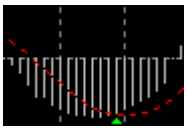
Year - returns the number of the current year.

Step-by-step creation of a simple Advisor

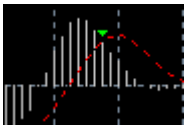
Let's try to create an Expert Advisor running on the standard MACD indicator, with the profit-taking capability, supporting trailing stops and using most safeguards for secure operation. In the example given below the trade is done by opening and controlling a single position.

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.



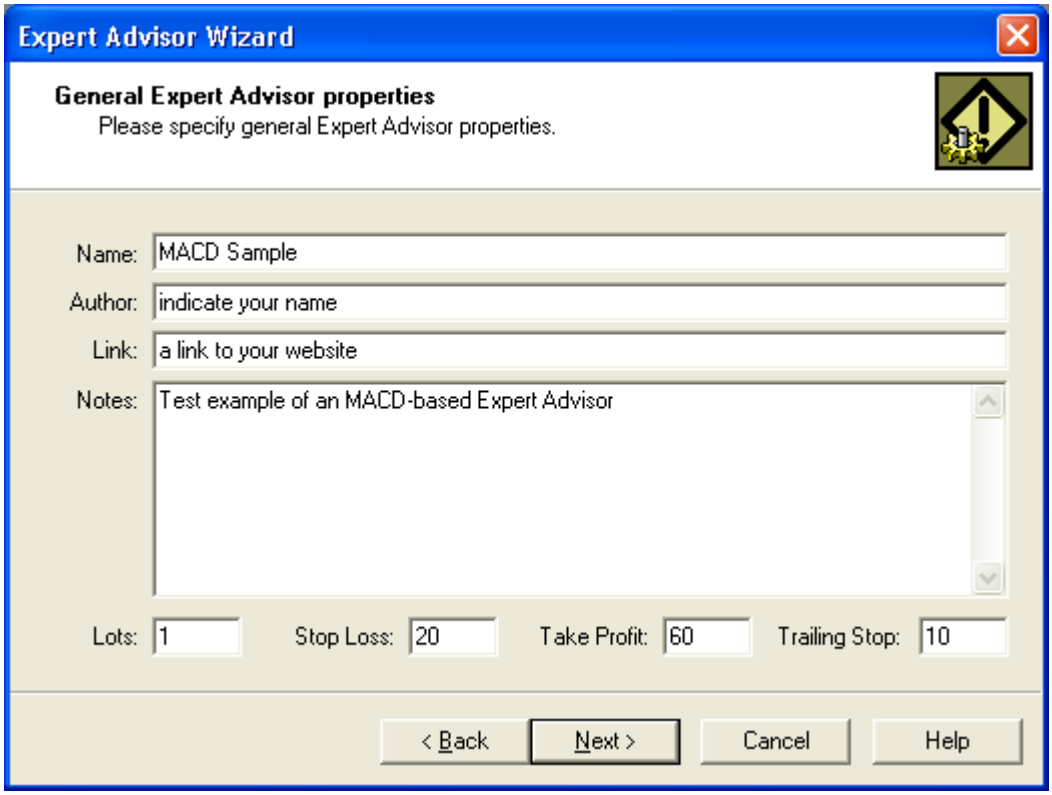
- **Long exit** - by execution of the take profit limit, by execution of the trailing stop or when MACD crosses its Signal Line (MACD is above zero, goes downwards and is crossed by the Signal Line going upwards).

- **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*. You can also indicate the default values you want to use for Lots, Stop Loss, Take Profit and Trailing Stop.



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 long 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 program 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,MODE_MAIN,0);     // MACD value on the current bar

   MacdPrevious=iMACD(12,26,9,MODE_MAIN,1);    // MACD value on the previous bar

   SignalCurrent=iMACD(12,26,9,MODE_SIGNAL,0); // Signal Line value on the current bar
   ```

SignalPrevious=iMACD(12,26,9,MODE_SIGNAL,1);// 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 MeraQuotes II language 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 program. 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 program.

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 'hillock'
  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 program text, to ensure greater flexibility. In the beginning of the program 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 program 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),B
id,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_OPEN
PRICE),
                    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<SignalPrevious and
Abs(MacdCurrent)>(MACDCloseLevel*Point) then
```

```
  {

CloseOrder(OrderValue(cnt,VAL_TICKET),OrderValue(cnt,VAL_LOTS),A
sk,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_OPEN
PRICE),


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 program

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, TrailingStop)
// 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, as
   // 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);
```

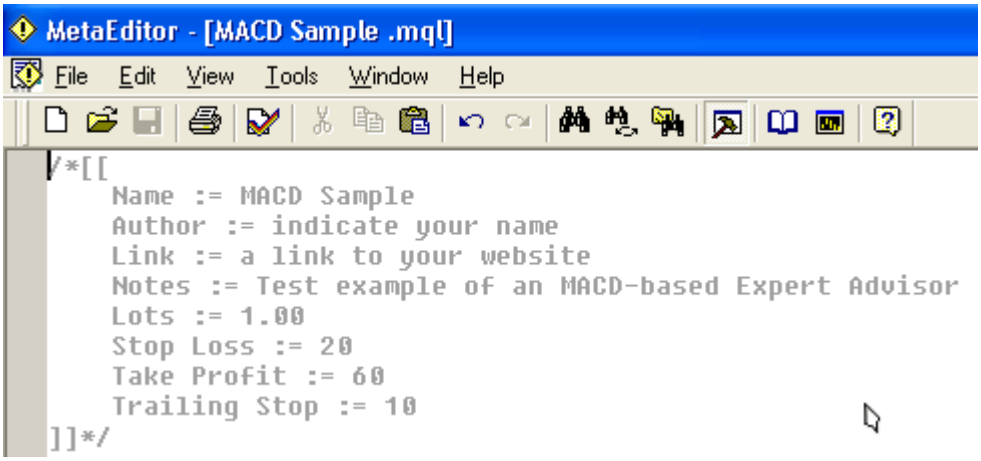
```
            Exit; // exiting
          };
        // we check - maybe, we already may or it's already time to set a trailing stop?
        If TrailingStop>0 then  // the user has put a trailing stop in his settings
         {                  // so, we set out to check it
          If (Bid-OrderValue(cnt,VAL_OPENPRICE))>(Point*TrailingStop) 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 long 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.
```

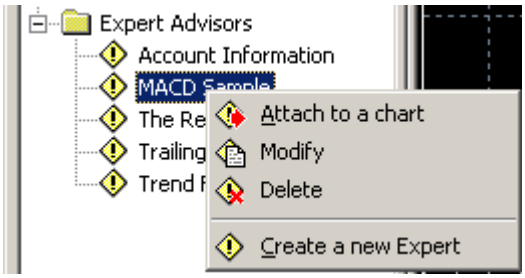
To complete setting up the Expert Advisor we need only to specify the values of the external

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 program printout against the grey background right into the MetaEditor), press the **Save** button to save the Expert Advisor.

Let's compile your Expert Advisor. In the MQL Editor click the Verify icon at the top - it looks like a sheet of paper with a checkmark.

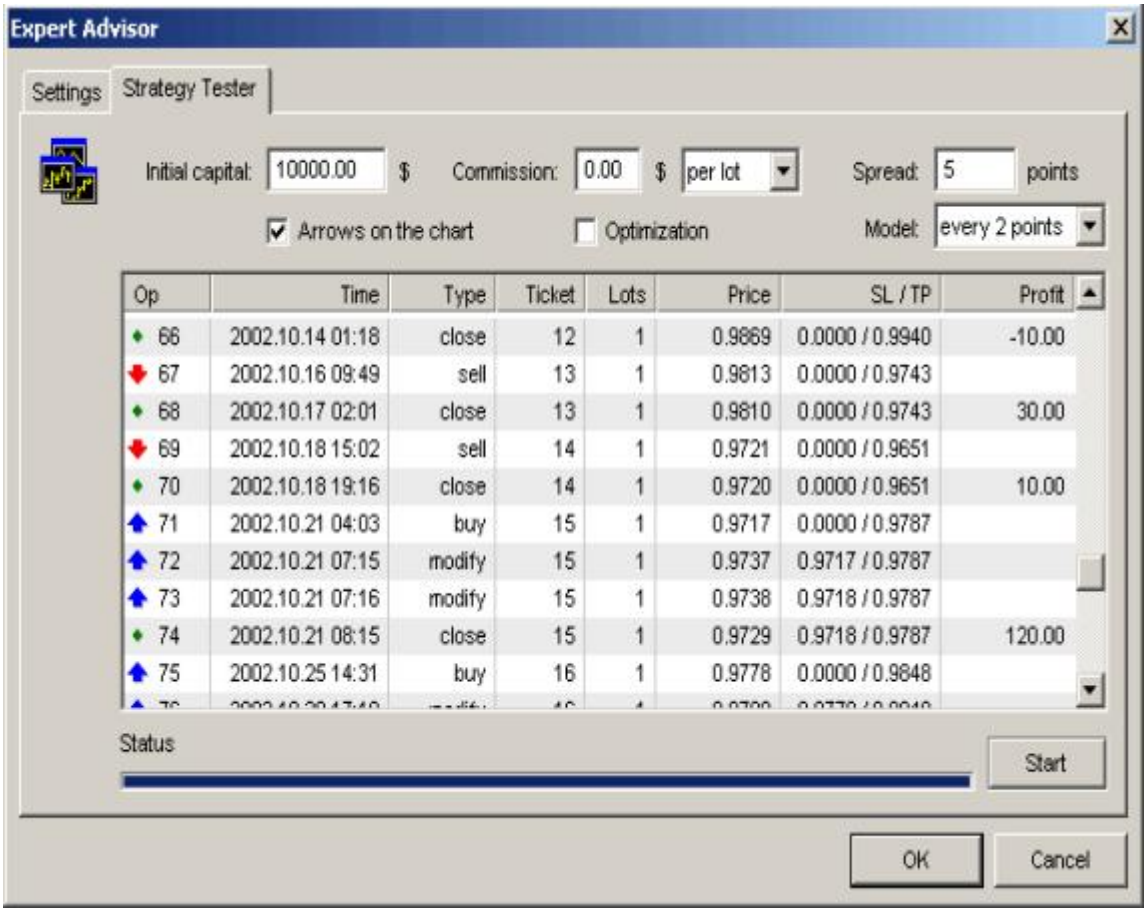


Step IV: testing the Expert Advisor on historical data



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:



: you need to keep in mind when writing Expert Advisors

Writing and testing Expert Advisors in the MetaTrader trading system has a number of particular features.

- Before opening a position, you need to check your account for availability of free funds. If the funds are insufficient, the position opening transaction will fail. It should be noted that the FreeMargin value has to be at least 1000 only for the testing purposes, since the testing price of one lot is 1000.

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

- After the opening, closing or modification of a position or after the deletion of a delayed order (upon execution of any of the following commands: SetOrder, CloseOrder, ModifyOrder or DeleteOrder) it is recommended to complete the operation of the Expert Advisor by the Exit statement, as upon the execution of the trade there is a 10-second trading timeout. It should be noted that this 10-second trading doesn't apply in the testing mode (you can execute several trades in a row), and, if the Expert Advisor operation is not completed with the Exit statement upon the execution of a trade, the testing results may differ from those of actual trading with the Expert Advisor.

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

Exit;          // exiting
```

To prevent the possibility of executing several trades with less than 10 seconds interval between them in the testing mode, you can simply make sure, before executing next trade, that at least 10 seconds passed since your last trade.

```
//making sure that the current time value is greater than 10 seconds since the execution of the last trade

If CurTime > LastTradeTime + 10 Then Begin

   SetOrder(OP_SELL,Lots,Bid,3,0,Bid-TakeProfit*Point,RED);          // executing

   Exit;

End;
```

- The historical data may be accessed using the indexed predefined variables Open, Close, High, Low, Volume. In this case the index represents the number of periods which have to be counted back.

```
// if the Close on the last bar is less than the Close on the last but one bar

If Close[1] < Close[2] Then Exit;
```

- Expert Advisor testing in the MetaTrader trading system supports 4 models.
    - **OHLC points** (Open/High/Low/Close). For the advisor testing purposes only the Open, High, Low, Close prices are used. This testing model is the fastest, however, the testing results may differ from the results of the actual trades executed by the Expert Advisor.
    - **Every 3 points**. When testing the Expert Advisor, a candlestick pattern with 3 points increment is modelled.
    - **Every 2 points**. When testing the Expert Advisor, a candlestick pattern with 2 points increment is modelled.
    - **Every 1 point**. When testing the Expert Advisor, a candlestick pattern with 1 points increment is modelled. This testing model is the slowest one, but the testing results are identical to the results of actual trade executed by the Expert Advisor (the 10-second interval between the trades is observed).
- In the writing and testing of an Expert Advisor, as well as in the testing of any other

programme, it is sometimes necessary to output some additional debugging information. MQL II language offers several possibilities for the output of such information.

- The Alert function sends to the screen a dialog box containing data defined by the user.
  Alert("Free margin is ", FreeMargin);

- The Comment function puts the data defined by the user into the left upper corner of the chart. The "\n" character sequence is used to break the line.
  Comment("Free margin is ", FreeMargin, "\nEquity is ", Equity);

- The Print function prints the data defined by the user into the system log.
  Print("Total trades are ", TotalTrades, "; Equity is ", Equity, "; Credit is ", Credit);

- The PrintTrade function outputs the details of the specified position into the log.
  PrintTrade(1);

- When the Expert Advisor is tested, the testing results are stored in a file with .log extension in the logs subdirectory of the directory you have installed the MetaTrader trading system in. If you often test your Expert Advisors, do not forget to periodically delete the log files, which may reach several megabytes in size.