MetaQuotes Language 4

MQL4 quick reference

About MetaQuotes Language 4

MetaQuotes Language 4 (MQL4) is a new built-in language for programming trading strategies. This language allows to create your own Expert Advisors that render the trade process management automatic and are perfectly suitable for implementing your own trade strategies. Also, with the help of MQL4 you can create your own Custom Indicators, Scripts and Libraries of functions.

A large number of functions necessary for the analysis of the current and past quotations, the basic arithmetic and logic operations are included in MQL4 structure. There are also basic indicators built in and commands of order placement and control.

The MetaEditor 4 text editor that highlights different constructions of MQL4 language is used for writing the program code. It helps users to orient in the expert system text quite easily. As an information book for MQL4 language we use MetaQuotes Language Dictionary. A brief guide contains functions divided into categories, operations, reserved words, and other language constructions and allows finding the description of every element we use.

Programs written in MetaQuotes Language 4 have different features and purposes:

- Expert Advisors is a mechanical trade system (MTS) linked up to a certain plot. The Advisor can not only inform you about a possibility to strike bargains, but also can make deals on the trade account automatically and direct them right to the trade server. Like most trade systems, the terminal supports testing strategies on historical data with displaying on the chart the spots where trades come in and out.
- Custom Indicators are an analogue of a technical indicator. In other words, Custom Indicators allow to create technical indicators in addition to those already integrated into client terminal. Like built-in indicators, they cannot make deals automatically and are aimed only at implementing analytical functions.
- Scripts are programs intended for single execution of some actions. Unlike Expert Advisors, Scripts are not run tick wise and have no access to indicator functions.
- Libraries are user functions libraries where frequently used blocks of user programs are stored.

Syntax

Format

Spaces, tabs, line feed/form feed symbols are used as separators. You can use any amount of such symbols instead of one. You should use tab symbols to enhance the readability of the text .

Comments

Multi line comments start with /* symbols and end with */ symbols. Such comments cannot be nested. Single comments start with // symbols, end with the symbol of a new line and can be nested into multi line comments. Comments are allowed where blank spaces are possible and tolerate any number of spaces.

Examples:

```
// single comment
/*  multi-
    line         // nested single comment
    comment
*/
```

Identifiers

Identifiers are used as names of variables, functions, and data types. The length of an identifier cannot exceed 31 characters.

Symbols you can use: the numbers 0-9, Latin capital and small letters a-z, A-Z (recognized as different symbols), the symbol of underlining (_). The first symbol cannot be a number. The identifier must not coincide with any reserved word.

Examples:

```
NAME1 namel Total_5 Paper
```

Reserved words

The identifiers listed below are fixed reserved words. A certain action is assigned to each of them, and they cannot be used for other purposes:

Data types	Memory classes	Operators	Other
bool	extern	break	false
color	static	case	true
datetime		continue	
double		default	
int		else	
string		for	
void		if	
		return	
		switch	
		while	

Data types

Data types overview

The main data types are:

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

We need the Color and Datetime types only to facilitate visualization and entering those parameters that we set from expert advisor property tab or custom indicator "Input parameters" tab. The data of Color and Datetime types are represented as integer values.
We use implicit type transformation. The priority of types at a transformation in ascending order is the following:

```
int  (bool,color,datetime);
double;
string;
```

Before operations (except for the assignment ones) are performed, the data have been transferred to the maximum precision type. Before assignment operations are performed, the data have been transferred to the integer type.

Integer constants

Decimal: numbers from 0 to 9; Zero should not be the first number.
Examples:

```
12, 111, -956 1007
```

Hexadecimal: numbers from 0 to 9, letters a-f or A-F to represent the values 10-15; they start with 0x or 0X.
Examples:

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

Integer constants can assume values from -2147483648 to 2147483647. If a constant exceeds this range, the result will not be defined.

Literal constants

Any single character enclosed in single quotes or a hexadecimal ASCII-code of a character looking like '\x10' is a character constant of integer type. Some characters like a single quote ('), double quote (") a question mark (?), a reverse slash (\) and control characters can be represented as a combination of characters starting with a reverse slash (\) according to the table below:

```
line feed                NL (LF)  \n
horizontal tab           HT       \t
carriage return          CR       \r
reverse slash            \        \\
single quote             '        \'
double quote             "        \"
hexadecimal ASCII-code   hh      \xhh
```

If a character different from those listed above follows the reverse slash, the result will not be defined.

Examples:
```
int a = 'A';
int b = '$';
int c = '©'; // code 0xA9
int d = '\xAE';   // symbol code ®
```

Boolean constants

Boolean constants may have the value of true or false, numeric representation of them is 1 or 0 respectively. We can also use synonyms True and TRUE, False and FALSE.
Examples:
```
bool a = true;
bool b = false;
bool c = 1;
```

Floating-point number constants

Floating-point constants consist of an integer part, a dot (.) and a fractional part. The integer and the fractional parts are a succession of decimal numbers. An unimportant fractional part with the dot can be absent.
Examples:
```
double a = 12.111;
double b = -956.1007;
double c = 0.0001;
double d = 16;
```
Floating-point constants can assume values from 2.2e-308 to 1.8e308. If a constant exceeds this range, the result will not be defined.

String constants

String constant is a succession of ASCII-code characters enclosed in double quotes: "Character constant".

A string constant is an array of characters enclosed in quotes. It is of the string type. Each string constant, even if it is identical to another string constant, is saved in a separate memory space. If you need to insert a double quote (") into the line, you must place a reverse slash (\) before it. You can insert any special character constants into the line if they have a reverse slash (\) before them. The length of a string constant lies between 0 and 255 characters. If the string constant is longer, the superfluous characters on the right are rejected.
Examples:
```
"This is a character string"
"Copyright symbol \t\xA9"
"this line with LF symbol \n"
"A" "1234567890" "0" "$"
```

Color constants

Color constants can be represented in three ways: by character representation; by integer representation; by name (for concrete Web colors only).
Character representation consists of four parts representing numerical rate values of three main color components - red, green, blue. The constant starts with the symbol C and is enclosed in single quotes. Numerical rate values of a color component lie in the range from 0 to 255.
Integer-valued representation is written in a form of hexadecimal or a decimal number. A hexadecimal number looks like 0x00BBGGRR where RR is the rate of the red color component, GG - of the green one and BB - of the blue one. Decimal constants are not directly reflected in RGB. They represent the decimal value of the hexadecimal integer representation.
Specific colors reflect the so-called Web colors set.
Examples:
```
// symbol constants
C'128,128,128'    // gray
```

```
C'0x00,0x00,0xFF' // blue
// named color
Red
Yellow
Black
// integer-valued representation
0xFFFFFF          // white
16777215          // white
0x008000          // green
32768             // green
```

Datetime constants

Datetime constants can be represented as a character line consisting of 6 parts for value of year, month, date, hour, minutes, and seconds. The constant is enclosed in simple quotes and starts with a D character.
Datetime constant can vary from Jan 1, 1970 to Dec 31, 2037.
Examples:

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

Operations & Expressions

Expressions

An expression consists of one or more operands and operation characters. An expression can be written in several lines.
Example:

```
a++; b = 10; x = (y*z)/w;
```

Note: An expression that ends with a semicolon is an operator.

Arithmetical operations

```
Sum of values                        i = j + 2;
Difference of values                 i = j - 3;
Changing the operation sign          x = - x;
Product of values                    z = 3 * x;
Division quotient                    i = j / 5;
Division remainder                   minutes = time % 60;
Adding 1 to the variable value       i++;
Subtracting 1 from the variable value   k--;
```

The operations of adding/subtracting 1 cannot be implemented in expressions.
Example:

```
int a=3;
a++;               // valid expression
int b=(a++)*3;     // invalid expression
```

The operation of assignment

Note: The value of the expression that includes this operation is the value of the left operand following the bind character.

```
Assigning the y value to the x variable                     y = x;
Adding x to the y variable                                  y += x;
Subtracting x from the y variable                           y -= x;
Multiplying the y variable by x                             y *= x;
Dividing the y variable by x                                y /= x;
Module x value of y                                         y %= x;
Logical shift of y representation to the right by x bit    y >>= x;
Logical shift of y representation to the left by x bit     y <<= x;
Bitwise operation AND                                       y &= x;
Bitwise operation OR                                        y |= x;
Bitwise operation exclusive OR                              y ^= x;
```

Note: There can be only one operation of assignment in the expression. You can implement bitwise operations with integer numbers only. The logical shift operation uses values of x less than 5 binary digits. The greater digits are rejected, so the shift is for the range of 0-31 bit. By %= operation a result sign is equal to the sign of divided number.

Operations of relation

The logical value false is represented with an integer zero value, while the logical value true is represented with any value differing from zero.
The value of expressions containing operations of relation or logical operations is 0 (false) or 1 (true).

```
True if a equals b                        a == b;
True if a does not equal b                a != b;
True if a is less than b                  a < b;
True if a is greater than b               a > b;
True if a is less than or equals b        a <= b;
True if a is greater than or equals b     a >= b;
```

Two unnormalized floating-point numbers cannot be linked by == or != operations. That is why it is necessary to subtract one from another, and the normalized outcome needs to be compared to null.

Boolean operations

The operand of negation NOT (!) must be of arithmetic type; the result equals 1 if the operand value is 0; the result equals 0 if the operand differs from 0.

```
// True if a is false.
if(!a)
  Print("not 'a'");
```

The logical operation OR (||) of values k and 1. The value k is checked first, the value 1 is checked only if k value is false. The value of this expression is true if the value of k or 1 is true.
Example:

```
if(x<k || x>l)
  Print("out of range");
```

The logical operation AND (&&) of values x and y. The value x is checked first; the value y is checked only if k value

is true. The value of this expression is true if the values of both x and y are true.
Example:

```
if(p!=x && p>y)
  Print("true");
n++;
```

Bitwise operations

One's complement of values of variables. The value of the expression contains 1 in all digits where n contains 0; the value of the expression contains 0 in all digits where n contains 1.

```
b = ~n;
```

Binary-coded representation of x is shifted to the right by y digits. The right shift is logical shift, that is the freed left-side bits will be filled with zeros.
Example:

```
x = x >> y;
```

The binary-coded representation of x is shifted to the right by y digits; the free digits on the right will be filled with zeroes.
Example:

```
x = x << y;
```

Bitwise operation AND of binary-coded x and y representations. The value of the expression contains 1 (true) in all digits where both x and y are not equal to zero; the value of the expression contains 0 (false) in all other digits.
Example:

```
b = ((x & y) != 0);
```

Bitwise operation OR of binary-coded x and y representations. The expression contains 1 in all digits where x and y not equals 0; the value of the expression contains 0 in all other digits.
Example:

```
b = x | y;
```

Bitwise operation EXCLUSIVE OR of binary-coded x and y representations. The expression contains 1 in all digits where x and y have different binary values; the value of the expression contains 0 in all other digits.
Example:

```
b = x ^ y;
```

Note: Bitwise operations are executed with integers only.

Other operations

Indexing. At addressing to i element of array, the value of the expression equals the value of the variable numbered as i.
Example:

```
array[i] = 3;
//Assign the value of 3 to array element with index i.
//Mind that the first array element
//is described with the expression array [0].
```

The call of function with x1,x2,...,xn arguments. The expression accepts the value returned by the function. If the returned value is of the void type, you cannot place such function call on the right in the assignment operation. Mind that the expressions x1,x2,...,xn are surely executed in this order.
Example:

```
double SL=Ask-25*Point;
double TP=Ask+25*Point;
int    ticket=OrderSend(Symbol(),OP_BUY,1,Ask,3,SL,TP,
                        "My comment",123,0,Red);
```

The "comma" operation is executed from left to right. A pair of expressions separated by a comma is calculated from left to right with a subsequent deletion of the left expression value. All side effects of left expression calculation can appear before we calculate the right expression. The result type and value coincide with the type and value of the right expression.

Precedence rules

Each group of operations in the table has the same priority. The higher the priority is, the higher the position of the group in the table is.
The execution order determines the grouping of operations and operands.

()	Function call	From left to right
[]	Array element selection	
!	Negation	From left to right
~	Bitwise negation	
-	Sign changing operation	
*	Multiplication	From left to right
/	Division	
%	Module division	
+	Addition	From left to right
-	Subtraction	
<<	Left shift	From left to right
>>	Right shift	
<	Less than	From left to right
<=	Less than or equals	
>	Greater than	
>=	Greater than or equals	
==	Equals	From left to right
!=	Not equal	
&	Bitwise AND operation	From left to right
^	Bitwise exclusive OR	From left to right
|	Bitwise OR operation	From left to right
&&	Logical AND	From left to right
||	Logical OR	From left to right
=	Assignment	From right to left
+=	Assignment addition	
-=	Assignment subtraction	
*=	Assignment multiplication	
/=	Assignment division	
%=	Assignment module	
>>=	Assignment right shift	
<<=	Assignment left shift	
&=	Assignment bitwise AND	
|=	Assignment bitwise OR	
^=	Assignment exclusive OR	
,	Comma	From left to right

Use parentheses to change the execution order of the operations.

Operators

Format and nesting
Compound operator
Expression operator
Break operator
Continue operator
Return operator
Conditional operator if
Conditional operator if-else
Switch operator
Cycle operator while
Cycle operator for

Format and nesting

Format. One operator can occupy one or several lines. Two or more operators can be located in the same line.
Nesting. Execution order control operators (if, if-else, switch, while and for) can be nested into each other.

Compound operator

A compound operator (a block) consists of one or more operators of any type enclosed in braces {}. The closing brace should not be followed by a semicolon (;).
Example:

```
if(x==0)
  {
   x=1; y=2; z=3;
  }
```

Expression operator

Any expression followed by a semicolon (;) is an operator. Here are some examples of expression operators:

Assignment operator.

```
   Identifier=expression;
```

Example:

```
x=3;
y=x=3; // error
```

You can use an assignment operator in an expression only once.

Function call operator

```
   Function_name(argument1,..., argumentN);
```

Example:

```
fclose(file);
```

Null operator

It consists of a semicolon (;) only. We use it to denote a null body of a control operator.

Break operator

A break; operator terminates the execution of the nearest nested outward operator switch, while or for. The control is given to the operator that follows the terminated one. One of the purposes of this operator is to finish the looping execution when a certain value is assigned to a variable.
Example:

```
// searching first zero element
for(i=0;i<array_size;i++)
  if((array[i]==0)
    break;
```

Continue operator

A continue; operator gives control to the beginning of the nearest outward cycle operator while or for, calling the next iteration. The purpose of this operator is opposite to that of break.
Example:

```
// summary of nonzero elements of array
int func(int array[])
  {
   int array_size=ArraySize(array);
   int sum=0;
   for(int i=0;i<array_size; i++)
     {
      if(a[i]==0) continue;
      sum+=a[i];
     }
   return(sum);
  }
```

Return operator

A return; operator terminates the current function execution and returns the control to the calling program.
A return(expression); operator terminates the current function execution and returns the control to the calling program together with the expression value. The operator expression is enclosed in parentheses. The expression should not contain an assignment operator.

Example:

```
return(x+y);
```

Conditional operator if

```
   if (expression)
     operator;
```

If the expression is true, the operator will be executed. If the expression is false, the control will be given to the expression following the operator.

Example:

```
if(a==x)
  temp*=3;
temp=MathAbs(temp);
```

Conditional operator if-else

```
   if (expression)
     operator1
   else
     operator2
```

If the expression is true, operator1 is executed and the control is given to the operator that follows operator2 (operator2 is not executed). If the expression is false, operator2 is executed.

The "else" part of the "if" operator can be omitted. Thus, a divergence may appear in nested "if" operators with an omitted "else" part. If it happens, "else" addresses to the nearest previous operator "if" in the block that has no "else" part.

Example:

```
//   The "else" part refers to the second "if" operator:
if(x>1)
   if(y==2)
      z=5;
   else
      z=6;
//   The "else" part refers to the first "if" operator:
if(x>l)
  {
   if(y==2) z=5;
  }
else
```

```
  {
   z=6;
  }
//   Nested operators
if(x=='a')
  {
   y=1;
  }
else if(x=='b')
  {
   y=2;
   z=3;
  }
else if(x=='c')
  {
   y = 4;
  }
else
  {
   Print("ERROR");
  }
```

Switch operator

```
switch (expression)
  {
   case constant1: operators; break;
   case constant2: operators; break;
   ...
   default: operators; break;
  }
```

It compares the expression value with constants in all variants of case and gives control to the operator that resembles the expression value. Each variant of the case can be marked with an integer or character constant or a constant expression. The constant expression must not include variables and function calls.
Example:

```
case 3+4: //valid
case X+Y: //invalid
```

Operators connected with a default label are executed if none of the constants in case operators equals the expression value. The default variant is not obligatory final. If none of the constants resembles the expression value and the default variant is absent, no actions are executed. The keyword case and the constant are just labels and if operators are executed for some variant of case the program will further execute the operators of all following variants until it hits a break operator. It allows linking one succession of operators with several variants. A constant expression is calculated during compilation.
None of two constants in one switch operator can have the same values.
Example:

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

```
     break;
  }
```

Cycle operator while

```
  while (expression)
    operator;
```

If the expression is true, the operator is executed till the expression becomes false. If the expression is false, the control will be given to the next operator.

Note: An expression value has been defined before the operator is executed. Therefore, if the expression is false from the very beginning, the operator is not executed at all.
Example:

```
while(k<n)
  {
   y=y*x;
   k++;
  }
```

Cycle operator for

```
  for (expression1; expression2; expression3)
   operator;
```

Expression1 describes the initialization of the cycle. Expression2 is the cycle termination check. If it is true, the loop body operator will be executed, Expression3 is executed. The cycle is repeated until Expression2 becomes false. If it is not false, the cycle is terminated, and the control is given to the next operator. Expression3 is calculated after each iteration. The 'for' operator is equivalent to the following succession of operators:

```
  expression1;
  while (expression2)
    {
     operator;
     expression3;
    };
```

Example:

```
for(x=1;x<=7;x++)
  Print(MathPower(x,2));
```

Any of the three or all three expressions can be absent in the FOR operator, but you should not omit the semicolons (;) that separate them.

If Expression2 is omitted, it is considered constantly true. The FOR (;;) operator is a continuous cycle equivalent to the WHILE(l) operator.
Each of the expressions 1 to 3 can consist of several expressions united by a comma operator ','.
Example:

```
//
for(i=0,j=n-l;i<n;i++,j--)
   a[i]=a[j];
```

Functions

Function definition

A function is defined by return value type declaration, by formal parameters and a compound operator (block) that describes actions the function executes.
Example:

```
double                                // type
linfunc (double x, double a, double b) // function name and
                                      // parameters list
  {
                                      // nested operators
   return (a*x + b);                  // returned value
  }
```

The "return" operator can return the value of the expression included into this operator. In case of a necessity, the expression value assumes the type of function result. A function that does not return a value must be of "void" type.
Example:

```
void errmesg(string s)
  {
   Print("error: "+s);
  }
```

Function call

function_name (x1,x2,...,xn)

Arguments (actual parameters) are transferred according to their value. Each expression x1,...,xn is calculated, and the value is passed to the function. The order of expressions calculation and the order of values loading are guaranteed. During the execution, the system checks the number and type of arguments given to the function. Such way of addressing to the function is called a value call. There is also another way: call by link. A function call is an expression that assumes the value returned by the function. This function type must correspond with the type of the returned value. The function can be declared or described in any part of the program:

```
int somefunc()
  {
   double a=linfunc(0.3, 10.5, 8);
  }
double linfunc(double x, double a, double b)
  {
   return (a*x + b);
  }
```

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

Any program begins its work with the "init()" function. "Init()" function attached to charts is launched also after client terminal has started and in case of changing financial symbol and/or charts periodicity.

Every program finishes its work with the "deinit()" function. "deinit()" function is launched also by client terminal shutdown, chart window closing, changing financial symbol and/or charts periodicity.

When new quotations are received, the "start()" function of attached expert advisors and custom indicator programs is executed. If, when receiving new quotations, the "start()" function triggered on the previous quotations was performed, the next calling "start()" function is executed only after "return()" instruction. All new quotations received during the program execution are ignored by the program.

Detaching of the program from charts, changing financial symbol and/or charts periodicity, charts closing and also client terminal exiting interrupts execution of program.

Execution of scripts does not depend on quotations coming.

Variables

Definitions

Definitions are used to define variables and to declare types of variables and functions defined somewhere else. A definition is not an operator. Variables must be declared before being used. Only constants can be used to initialize variables.

The basic types are:

- string - a string of characters;
- int - an integer;
- double - a floating-point number (double precision);
- bool - a boolean number "true" or "false".

Example:

```
string MessageBox;
int    Orders;
double SymbolPrice;
bool   bLog;
```

The additional types are:

- datetime is date and time, unsigned integer, containing seconds since 0 o'clock on January, 1, 1970.
- color - integer reflecting a collection of three color components.

The additional data types make sense only at the declaration of input data for more convenient their representation in a property sheet.
Example:

```
extern datetime tBegin_Data   = D'2004.01.01 00:00';
extern color    cModify_Color = C'0x44,0xB9,0xE6';
```

Arrays
Array is the indexed sequence of the identical-type data.
Example:

```
int    a[50];       //A one-dimensional array of 50 integers.
double m[7][50];    //Two-dimensional array of seven arrays,
                    //each of them consisting of 50 integers.
```

Only an integer can be an array index. No more than four-dimensional arrays can be declared.

Defining local variables

The variable declared inside any function is local. The scope of a local variable is limited to limits of the function inside which it is declared. The local variable can be initialized by outcome of any expression. Every call of function execute the initialization of local variables. Local variables are stored in memory area of corresponding function.

Formal parameters
Examples:

```
void func(int x, double y, bool z)
  {
   ...
  }
```

Formal parameters are local. Scope is the block of the function. Formal parameters must have names differing from those of external variables and local variables defined within one function. In the block of the function to the formal parameters some values can be assigned. Formal parameters can be initialized by constants. In this case, the initializing value is considered as a default value. The parameters following the initialized parameter should be initialized, as well.

By calling this function the initialized parameters can be omitted, instead of them defaults will be substituted.

Example:

```
func(123, 0.5);
```

Parameters are passed by value. These are modifications of a corresponding local variable inside the called function will not be reflected in any way in the calling function. It is possible to pass arrays as parameters. However, for an array passed as parameter, it is impossible to change the array elements.

There is a possibility to pass parameters by reference. In this case, modification of such parameters will be reflected on corresponded variables in the called function. To point, that the parameter is passed by reference, after a data type, it is necessary to put the modifier &.

Example:

```
void func(int& x, double& y, double& z[])
  {
   ...
  }
```

Arrays also can be passed by reference, all modifications will be reflected in the initial array. The parameters that passed by reference, cannot be initialized by default values.

Static variables

The memory class "static" defines a static variable. The specifier "static" is declared before a data type.
Example:

```
  {
   static int flag
  }
```

Static variables are constant ones since their values are not lost when the function is exited. Any variables in a block, except the formal parameters of the function, can be defined as static. The static variable can be initialized by corresponded type constant, as against a simple local variable which can be initialized by any expression. If there is no explicit initialization, the static variable is initialized with zero. Static variables are initialized only once before calling "init()" function. That is at exit from the function inside which the static variable is declared, the value of this variable being not lost.

Defining global variables

They are defined on the same level as functions, i.e. they are not local in any block.
Example:

```
int Global_flag;
int start()
  {
   ...
  }
```

Scope of global variables is the whole program. Global variables are accessible from all functions defined in the program. They are initialized with zero if no other initial value is explicitly defined. The global variable can be initialized only by corresponded type constant. Initialization of global variables is made only once before execution of "init()" function.

Note: it is not necessary to confuse the variables declared at a global level, to global variables of Client Terminal, access to which is carried out by GlobalVariable...() function.

Defining extern variables

The memory class "extern" defines an extern variable. The specifier "extern" is declared before a data type.
Example:

```
extern double InputParameter1 = 1.0;
int init()
```

```
  {
   ...
  }
```

Extern variables define input data of the program, they are accessible from a property program sheet. It is not meaningful to define extern variables in scripts. Arrays cannot represent itself as extern variables.

Initializing variables

Any variable can be initialized during its definition. Any permanently located variable is initialized with zero (0) if no other initial value is explicitly defined. Global and static variables can be initialized only by constant of corresponded type. Local variables can be initialized by any expression, and not just a constant. Initialization of global and static variables is made only once. Initialization of local variables is made each time by call of corresponded functions.

Basic types

Examples:

```
int mt = 1;             // integer initialization
// initialization floating-point number (double precision)
double p = MarketInfo(Symbol(),MODE_POINT);
// string initialization
string s = "hello";
```

Arrays

Example:

```
int mta[6] = {1,4,9,16,25,36};
```

The list of array elements must be enclosed by curly braces. If the array size is defined, the values being not explicitly defined equal 0.

External function definition

The type of external functions defined in another component of a program must be explicitly defined. The absence of such a definition may result in errors during the compilation, assembling or execution of your program. While describing an external object, use the key word #import with the reference to the module.

Examples:

```
#import "user32.dll"
  int     MessageBoxA(int hWnd ,string szText,
                      string szCaption,int nType);
  int     SendMessageA(int hWnd,int Msg,int wParam,int lParam);
#import "lib.ex4"
  double  round(double value);
#import
```

Preprocessor

- **Declaring of constant**
- **Controlling compilation**
- **Including files**
- **Importing functions and other modules**

Declaring of constant

If the first character in a program line is #, it means that this line is a compiler command. Such a compiler command ends with a carriage-return character.

#define identifier_value

The identifier of a constant obeys the same rules as variable names. The value can be of any type. Example:

```
#define ABC          100
```

```
#define PI           0.314
#define COMPANY_NAME "MetaQuotes Software Corp."
```

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

Controlling compilation

#property identifier_value

The list of predefined constant identifiers. Example:

```
#property link        "http://www.metaquotes.net"
#property copyright   "MetaQuotes Software Corp."
#property stacksize   1024
```

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

The compiler will write the declared values to the settings of the executable module.

Including files

Note: The #include command line can be placed anywhere in the program, but usually all inclusions are placed at the beginning of the source code.

#include <file_name>

Example:

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

The preprocessor replaces this line with the content of the file win32.h. Angle brackets mean that the file win32.h will be taken from the default directory (usually, terminal_directory\experts\include). The current directory is not searched.

#include "file_name"

Example:

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

The compiler replaces this line with the content of the file mylib.h. Since this name is enclosed in quotes, the search is performed in the current directory (where the main file of the source code is located). If the file is not found in the current directory, the error will be messaged.

Importing functions and other modules

```
#import "file_name"
 func1();
 func2();
```

#import

Example:

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

Functions are imported from MQL4 compiled modules (*.ex4 files) and from operating system modules (*.dll files). In the latter case, the imported functions are also declared. A new #import command (it can be without parameters) finishes the description of imported functions.

Account Information

double **AccountBalance()**

Returns balance value of the current account.

Sample

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

double **AccountCredit()**

Returns credit value of the current account.

Sample

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

string **AccountCompany()**

Returns the current account company name.

Sample

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

string **AccountCurrency()**

Returns currency name of the current account.

Sample

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

double AccountEquity()

Returns equity value of the current account.

Sample

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

double AccountFreeMargin()

Returns free margin value of the current account.

Sample

```
Print("Account free margin = ",AccountFreeMargin());
```

int AccountLeverage()

Returns leverage of the current account.

Sample

```
Print("Account #",AccountNumber(), " leverage is ", AccountLeverage());
```

double AccountMargin()

Returns margin value of the current account.

Sample

```
Print("Account margin ", AccountMargin());
```

string AccountName()

Returns the current account name.

Sample

```
Print("Account name ", AccountName());
```

int AccountNumber()

Returns the number of the current account.

Sample

```
Print("account number ", AccountNumber());
```

double AccountProfit()

Returns profit value of the current account .

Sample

```
Print("Account profit ", AccountProfit());
```

Array Functions

int ArrayBsearch(**double array[], double value, int count=WHOLE_ARRAY, int start=0, int direction=MODE_ASCEND)**

Returns the index of the first occurrence of a value in the first dimension of array if possible, or the nearest one, if the occurrence is not found.
The function cannot be used with string arrays and serial numeric arrays.
Note: Binary search processes only sorted arrays. To sort numeric arrays use ArraySort() functions.

Parameters

array[]	-	The numeric array to search for.
value	-	The value to search for.
count	-	Count of elements to search for. By default, it searches in the whole array.
start	-	Starting index to search for. By default, the search starts on the first element.
direction	-	Search direction. It can be any of the following values: MODE_ASCEND searching in forward direction, MODE_DESCEND searching in the backward direction.

Sample

```
datetime daytimes[];
int      shift=10,dayshift;
// All the Time[] timeseries are sorted in descendant mode
ArrayCopySeries(daytimes,MODE_TIME,Symbol(),PERIOD_D1);
if(Time[shift]>=daytimes[0]) dayshift=0;
else
  {
   dayshift=ArrayBsearch(daytimes,Time[shift],WHOLE_ARRAY,0,MODE_DESCEND);
   if(Period()<PERIOD_D1) dayshift++;
  }
Print(TimeToStr(Time[shift])," corresponds to ",dayshift," day bar opened at ",
      TimeToStr(daytimes[dayshift]));
```

int ArrayCopy(**object& dest[], object source[], int start_dest=0, int start_source=0, int count=WHOLE_ARRAY)**

Copies an array to another one. Arrays must be of the same type, but arrays with type double[], int[], datetime[], color[], and bool[] can be copied as arrays with same type.
Returns the amount of copied elements.

Parameters

dest[]	-	Destination array.
source[]	-	Source array.
start_dest	-	Starting index for the destination array. By default, start index is 0.
start_source	-	Starting index for the source array. By default, start index is 0.
count	-	The count of elements that should be copied. By default, it is WHOLE_ARRAY constant.

Sample

```
double array1[][6];
double array2[10][6];
// fill array with some data
ArrayCopyRates(array1);
ArrayCopy(array2, array1,0,Bars-9,10);
// now array2 has first 10 bars in the history
```

int ArrayCopyRates(double& dest_array[], string symbol=NULL, int timeframe=0)

Copies rates to the two-dimensional array from chart RateInfo array, where second dimension has 6 elements:
0 - time,
1 - open,
2 - low,
3 - high,
4 - close,
5 - volume.
Note: Usually retrieved array used to pass large blocks of data to the DLL functions.

Parameters

dest_array[]	-	Reference to the two-dimensional destination numeric array.
symbol	-	symbol name, by default, current chart symbol name is used.
timeframe	-	Time frame, by default, the current chart time frame is used. It can be any of Time frame enumeration values.

Sample

```
double array1[][6];
ArrayCopyRates(array1,"EURUSD", PERIOD_H1);
Print("Current bar ",TimeToStr(array1[0][0]),"Open", array1[0][1]);
```

int ArrayCopySeries(double& array[], int series_index, string symbol=NULL, int timeframe=0)

Copies a series array to another one and returns the count of copied elements.
Note: If series_index is MODE_TIME, the first parameter must be a datetime array.

Parameters

array[]	-	Reference to the destination one-dimensional numeric array.
series_index	-	Series array identifier. It can be any of Series array identifiers enumeration values.
symbol	-	Symbol name, by default, the current chart symbol name is used.
timeframe	-	Time frame, by default, the current chart time frame is used. It can be any of Time frame enumeration values.

Sample

```
  datetime daytimes[];
  int      shift=10,dayshift;
  // All the Time[] timeseries are sorted in descendant mode
  ArrayCopySeries(daytimes,MODE_TIME,Symbol(),PERIOD_D1);
  if(Time[shift]>=daytimes[0]) dayshift=0;
  else
    {
     dayshift=ArrayBsearch(daytimes,Time[shift],WHOLE_ARRAY,0,MODE_DESCEND);
     if(Period()<PERIOD_D1) dayshift++;
    }
  Print(TimeToStr(Time[shift])," corresponds to ",dayshift," day bar opened at ",
TimeToStr(daytimes[dayshift]));
```

int ArrayDimension(int array[])

Returns array rank (dimensions count).

Parameters

array[]	-	array to retrieve dimensions count.

Sample

```
int num_array[10][5];
int dim_size;
dim_size=ArrayDimension(num_array);
// dim_size is 2
```

bool ArrayGetAsSeries(object array[])

Returns true if array is organized as a series array (array elements indexed from last to first) otherwise return false.

Parameters

array[]	-	Array to check.

Sample

```
if(ArrayGetAsSeries(array1)==true)
  Print("array1 is indexed as a series array");
else
  Print("array1 is indexed normally (from left to right)");
```

int ArrayInitialize(double& array[], double value)

Sets all elements of numeric array to the same value. Returns the count of initialized element.
Note: It is useless to initialize index buffers in the custom indicator *init()* function.

Parameters

array[]	-	Numeric array to be initialized.
value	-	New value to be set.

Sample

```
//---- setting all elements of array to 2.1
double myarray[10];
ArrayInitialize(myarray,2.1);
```

bool ArrayIsSeries(object array[])

Returns true if the array checked is a series array (time,open,close,high,low, or volume).

Parameters

array[]	-	Array to check.

Sample

```
if(ArrayIsSeries(array1)==false)
  ArrayInitialize(array1,0);
else
  {
   Print("Series array cannot be initialized!");
   return(-1);
  }
```

int ArrayMaximum(double array[], int count=WHOLE_ARRAY, int start=0)

Searches for elements with maximum value and returns its position.

Parameters

array[]	-	The numeric array to search for.
count	-	Scans for the count of elements in the array.
start	-	Start searching on the start index.

Sample

```
double num_array[15]={4,1,6,3,9,4,1,6,3,9,4,1,6,3,9};
int    maxValueIdx=ArrayMaximum(num_array);
Print("Max value = ", num_array[maxValueIdx]);
```

int ArrayMinimum(double array[], int count=WHOLE_ARRAY, int start=0)

Searches for element with minimum value and returns its position.

Parameters

array[] - The numeric array to search for.
count - Scans for the count of elements in the array.
start - Start searching on the start index.

Sample

```
double num_array[15]={4,1,6,3,9,4,1,6,3,9,4,1,6,3,9};
double minValueidx=ArrayMinimum(num_array);
Print("Min value = ", num_array[minValueIdx]);
```

int ArrayRange(object array[], int range_index)

Returns the count of elements in the indicated dimension of the array. Since indexes are zero-based, the size of dimension is 1 greater than the largest index.

Parameters

array[] - Array to check
range_index - Dimension index.

Sample

```
int    dim_size;
double num_array[10,10,10];
dim_size=ArrayRange(num_array, 1);
```

int ArrayResize(object& array[], int new_size)

Sets new size to the first dimension. If success returns count of all elements contained in the array after resizing, otherwise, returns zero and array is not resized.

Parameters

array[] - Array to resize.
new_size - New size for the first dimension.

Sample

```
double array1[][4];
int    element_count=ArrayResize(array, 20);
// element count is 80 elements
```

bool ArraySetAsSeries(double& array[], bool set)

Sets indexing order of the array like a series arrays, i.e. last element has zero index. Returns previous state.

Parameters

array[] - The numeric array to set.
set - The Series flag to set (true) or drop (false).

Sample

```
  double macd_buffer[300];
  double signal_buffer[300];
  int    i,limit=ArraySize(macd_buffer);
  ArraySetAsSeries(macd_buffer,true);
  for(i=0; i<limit; i++)
     macd_buffer[i]=iMA(NULL,0,12,0,MODE_EMA,PRICE_CLOSE,i)-
iMA(NULL,0,26,0,MODE_EMA,PRICE_CLOSE,i);
  for(i=0; i<limit; i++)
     signal_buffer[i]=iMAOnArray(macd_buffer,limit,9,0,MODE_SMA,i);
```

int ArraySize(object array[])

Returns the count of elements contained in the array.

Parameters

array[]	-	Array of any type.

Sample

```
int count=ArraySize(array1);
for(int i=0; i<count; i++)
  {
   // do some calculations.
  }
```

```
int ArraySize(   double& array[], int count=WHOLE_ARRAY, int start=0,
                 int sort_dir=MODE_ASCEND)
```

int ArraySort(double& array[], int count=WHOLE_ARRAY, int start=0, int sort_dir=MODE_ASCEND)

Sorts numeric arrays by first dimension. Series arrays cannot be sorted by ArraySort().

Parameters

array[]	-	The numeric array to sort.
count	-	Count of elements to sort.
start	-	Starting index.
sort_dir	-	Array sorting direction. It can be any of the following values: MODE_ASCEND - sort ascending, MODE_DESCEND - sort descending.

Sample

```
double num_array[5]={4,1,6,3,9};
// now array contains values 4,1,6,3,9
ArraySort(num_array);
// now array is sorted 1,3,4,6,9
ArraySort(num_array,MODE_DESCEND);
// now array is sorted 9,6,4,3,1
```

Common functions

Alert()
ClientTerminalName()
CompanyName()
Comment()
GetLastError()
GetTickCount()
HideTestIndicators()
IsConnected()
IsDemo()
IsDllsAllowed()
IsLibrariesAllowed()
IsStopped()
IsTesting()
IsTradeAllowed()
MarketInfo()
MessageBox()
Period()
PlaySound()
Print()
RefreshRates()
SendMail()
ServerAddress()
Sleep()
SpeechText()
Symbol()
UninitializeReason()

void Alert(...)

Displays a dialog box containing the user-defined data. Parameters can be of any type. Arrays cannot be passed to the Alert function. Data of double type printed with 4 decimal digits after point. To print with more precision use DoubleToStr() function. Data of bool, datetime and color types will be printed as its numeric presentation. To print values of datetime type as string convert it by TimeToStr() function.

See also: Comment() and Print() functions.

Parameters

... - Any values, separated by commas.

Sample

```
 if(Close[0]>SignalLevel)
   Alert("Close price coming ", Close[0],"!!!");
```

string ClientTerminalName()

Returns Client Terminal Name.

Sample

```
 Print("Terminal name is ",ClientTerminalName());
```

string CompanyName()

Returns Company name

Sample

```
 Print("Company name is ",CompanyName());
```

void Comment(...)

Prints some message to the left top corner of the chart. Parameters can be of any type. Arrays cannot be passed to the Comment() function. Arrays should be output elementwise. Data of double type printed with 4 decimal digits after point. To print with more precision use DoubleToStr() function. Data of bool, datetime and color types will be printed as its numeric presentation. To print values of datetime type as string convert it by TimeToStr() function.
See also: Alert() and Print() functions.

Parameters

... - Any values, separated by commas.

Sample

```
  double free=AccountFreeMargin();
  Comment("Account free margin is ",DoubleToStr(free,2),"\n","Current time is
",TimeToStr(CurTime()));
```

int GetLastError()

Returns last occurred error after an operation and sets internal last error value to zero.

Sample

```
  int err;
  int handle=FileOpen("somefile.dat", FILE_READ|FILE_BIN);
  if(handle<1)
    {
     err=GetLastError();
     Print("error(",err,"): ",ErrorDescription(err));
     return(0);
    }
```

int GetTickCount()

The GetTickCount() function retrieves the number of milliseconds that have elapsed since the system was started. It is limited to the resolution of the system timer.

Sample

```
int start=GetTickCount();
// do some hard calculation...
Print("Calculation time is ", GetTickCount()-start, " milliseconds.");
```

void HideTestIndicators(bool hide)

The function sets a flag hiding indicators called by the Expert Advisor. After the chart has been tested and opened the flagged indicators will not be drawn on the testing chart. Every indicator called will first be flagged with the current hiding flag.

Parameters

hide	-	TRUE - if indicators must be hidden, otherwise, FALSE.

Sample

```
HideTestIndicators(true);
```

bool IsConnected()

Returns true if client terminal has opened connection to the server, otherwise returns false.

Sample

```
if(!IsConnected())
  {
   Print("Connection is broken!");
   return(0);
  }
// Expert body that need opened connection
// ...
```

bool IsDemo()

Returns true if expert runs on demo account, otherwise returns false.

Sample

```
if(IsDemo()) Print("I am working on demo account");
else Print("I am working on real account");
```

bool IsDllsAllowed()

Returns true if DLL function call is allowed for the expert, otherwise returns false. **See also** IsLibrariesAllowed(), IsTradeAllowed().

Sample

```
#import "user32.dll"
   int     MessageBoxA(int hWnd ,string szText, string szCaption,int nType);
...
...
if(IsDllsAllowed()==false)
  {
   Print("DLL call is not allowed. Experts cannot run.");
   return(0);
  }
// expert body that calls external DLL functions
MessageBoxA(0,"an message","Message",MB_OK);
```

bool IsLibrariesAllowed()

Returns true if expert can call library function, otherwise returns false. **See also** IsDllsAllowed(), IsTradeAllowed().

Sample

```
#import "somelibrary.ex4"
   int somefunc();
...
...
if(IsLibrariesAllowed()==false)
  {
   Print("Library call is not allowed. Experts cannot run.");
   return(0);
  }
// expert body that calls external DLL functions
somefunc();
```

bool IsStopped()

Returns true if expert in the stopping state, otherwise returns false. This function can be used in the cycles to determine expert unloading.

Sample

```
while(expr!=false)
  {
   if(IsStopped()==true) return(0);
   // long time procesing cycle
   // ...
  }
```

bool IsTesting()

Returns true if expert runs in the testing mode, otherwise returns false.

Sample

```
if(IsTesting()) Print("I am testing now");
```

bool IsTradeAllowed()

Returns true if trade is allowed for the expert, otherwise returns false. **See also** IsDllsAllowed(), IsLibrariesAllowed().

Sample

```
if(IsTradeAllowed()) Print("Trade allowed");
```

double MarketInfo(string symbol, int type)

Returns value from Market watch window.

Parameters

symbol	-	Instrument symbol.
type	-	Returning data type index. It can be any of Market information identifiers value.

Sample

```
double var;
var=MarketInfo("EURUSD",MODE_BID);
```

int MessageBox(string text=NULL, string caption=NULL, int flags=EMPTY)

The MessageBox function creates, displays, and operates a message box. The message box contains an application-defined message and title, plus any combination of predefined icons and push buttons. If the function succeeds, the return value is one of the MessageBox return code values.

Parameters

text	-	Optional text that contains the message to be displayed.
caption	-	Optional text that contains the dialog box title.If this parameter is NULL, the title will be name of expert.
flags	-	Specifies the contents and behavior of the dialog box.This optional parameter can be a combination of flags from the following groups of flags.

Sample

```
 #include <WinUser32.mqh>

 if(ObjectCreate("text_object", OBJ_TEXT, 0, D'2004.02.20 12:30', 1.0045)==false)
   {
    int ret=MessageBox("ObjectCreate() fails with code
"+GetLastError()+"\nContinue?", "Question", MB_YESNO|MB_ICONQUESTION);
    if(ret==IDNO) return(false);
   }
 // continue
```

int Period()

Returns the number of minutes defining the used period (chart timeframe).

Sample

```
 Print("Period is ", Period());
```

void PlaySound(string filename)

Function plays sound file. File must be located at the `terminal_dir\sounds` directory or its subdirectory.

Parameters

filename	-	Sound file name.

Sample

```
 if(IsDemo()) PlaySound("alert.wav");
```

void Print(...)

Prints some message to the experts log. Parameters can be of any type. Arrays cannot be passed to the Print() function. Arrays should be printed elementwise. Data of double type printed with 4 decimal digits after point. To print with more precision use DoubleToStr() function. Data of bool, datetime and color types will be printed as its numeric presentation. To print values of datetime type as string convert it by TimeToStr() function.
See also: Alert() and Comment() functions.

Parameters

...	-	Any values, separated by commas.

Sample

```
 Print("Account free margin is ", AccountFreeMargin());
 Print("Current time is ", TimeToStr(CurTime()));
 double pi=3.141592653589793;
 Print("PI number is ", DoubleToStr(pi,8));
 // Output: PI number is 3.14159265
 // Array printing
 for(int i=0;i<10;i++)
   Print(Close[i]);
```

bool RefreshRates()

Refreshing data in the built-in variables and series arrays. This function is used when expert advisor calculates for a long time and needs refreshing data. Returns true if data are refreshed, otherwise false.

Sample

```
  int ticket;
  while(true)
    {
     ticket=OrderSend(Symbol(),OP_BUY,1.0,Ask,3,0,0,"expert
comment",255,0,CLR_NONE);
     if(ticket<=0)
       {
        int error=GetLastError();
        if(error==134) break;            // not enough money
        if(error==135) RefreshRates();   // prices changed
        break;
       }
     else { OrderPrint(); break; }
     //---- 10 seconds wait
     Sleep(10000);
    }
```

void SendMail(string subject, string some_text)

Sends mail to address set in the Tools->Options->EMail tab if enabled. **Note:** Posting e-mail can be denied or address can be empty.

Parameters

subject	-	Subject text.
some_text	-	Mail body.

Sample

```
 double lastclose=Close[0];
 if(lastclose<my_signal)
   SendMail("from your expert", "Price dropped down to "+DoubleToStr(lastclose));
```

string ServerAddress()

Returns connected server address in form of a text string.

Sample

```
 Print("Server address is ", ServerAddress());
```

void Sleep(int milliseconds)

The Sleep function suspends the execution of the current expert for a specified interval.

Parameters

milliseconds	-	Sleeping interval in milliseconds.

Sample

```
  Sleep(5);
```

void SpeechText(string text, int lang_mode=SPEECH_ENGLISH)

Computer speaks some text.

Parameters

text	-	Speaking text.
lang_mode	-	SPEECH_ENGLISH (by default) or SPEECH_NATIVE values.

Sample

```
 double lastclose=Close[0];
 SpeechText("Price dropped down to "+DoubleToStr(lastclose));
```

string Symbol()

Returns a text string with the name of the current financial instrument.

Sample

```
  int total=OrdersTotal();
  for(int pos=0;pos<total;pos++)
    {
     // check selection result becouse order may be closed or deleted at this time!
     if(OrderSelect(pos, SELECT_BY_POS)==false) continue;
     if(OrderType()>OP_SELL || OrderSymbol()!=Symbol()) continue;
     // do some orders processing...
    }
```

int UninitializeReason()

Returns the code of the uninitialization reason for the experts, custom indicators, and scripts. Return values can be one of Uninitialize reason codes.

Sample

```
  // this is example
  int deinit()
    {
     switch(UninitializeReason())
       {
        case REASON_CHARTCLOSE:
        case REASON_REMOVE:      CleanUp(); break; // clean up and free all expert's
resources.
        case REASON_RECOMPILE:
        case REASON_CHARTCHANGE:
        case REASON_PARAMETERS:
        case REASON_ACCOUNT:     StoreData(); break;  // prepare to restart
       }
     //...
    }
```

Conversion functions

- **CharToStr()**
- **DoubleToStr()**
- **NormalizeDouble()**
- **StrToDouble()**
- **StrToInteger()**
- **StrToTime()**
- **TimeToStr()**

string CharToStr(int char_code)

Returns string with one symbol that have specified code

Parameters

char_code	-	ASCII char code.

Sample

```
  string str="WORL" + CharToStr(44); // 44 is code for 'D'
  // resulting string will be WORLD
```

string DoubleToStr(double value, int digits)

Returns text string with the specified numerical value transformed into the specified precision format.

Parameters

value - Numerical value.
digits - Precision format, number of digits after decimal point (0-8).

Sample

```
string value=DoubleToStr(1.28473418, 5);
// value is 1.28473
```

double NormalizeDouble(double value, int digits)

Rounds floating point number to specified decimal places.

Parameters

value - Floating point value.
digits - Precision format, number of digits after decimal point (0-8).

Sample

```
double var1=0.123456789;
Print(NormalizeDouble(var1,5));
// output: 0.12346
```

double StrToDouble(string value)

Converts string representation of number to type double.

Parameters

value - String containing value in fixed number format.

Sample

```
double var=StrToDouble("103.2812");
```

int StrToInteger(string value)

Converts string representation of number to type integer.

Parameters

value - String containing integer number.

Sample

```
int var1=StrToInteger("1024");
```

datetime StrToTime(string value)

Converts string in the format "yyyy.mm.dd hh:mi" to type datetime.

Parameters

value - String value of date/time format such as "yyyy.mm.dd hh:mi".

Sample

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

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

Returns time as string in the format "yyyy.mm.dd hh:mi".

Parameters

value	-	Positive number of seconds from 00:00 January 1, 1970.
mode	-	Optional data output mode can be one or combination of: TIME_DATE get result in form "yyyy.mm.dd", TIME_MINUTES get result in form "hh:mi", TIME_SECONDS get result in form "hh:mi:ss".

Sample

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

Custom Indicator functions

void IndicatorBuffers(int count)

Allocates memory for buffers used for custom indicator calculations. Cannot be greater than 8 and less than indicator_buffers property. If custom indicator requires additional buffers for counting then use this function for pointing common buffers count.

Parameters

count	-	Buffers count to allocate. Should be up to 8 buffers.

Sample

```
#property  indicator_separate_window
#property  indicator_buffers 1
#property  indicator_color1  Silver
//---- indicator parameters
extern int FastEMA=12;
extern int SlowEMA=26;
extern int SignalSMA=9;
//---- indicator buffers
double     ind_buffer1[];
double     ind_buffer2[];
double     ind_buffer3[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int init()
  {
//---- 2 additional buffers are used for counting.
   IndicatorBuffers(3);
//---- drawing settings
   SetIndexStyle(0,DRAW_HISTOGRAM,STYLE_SOLID,3);
   SetIndexDrawBegin(0,SignalSMA);
   IndicatorDigits(MarketInfo(Symbol(),MODE_DIGITS)+2);
//---- 3 indicator buffers mapping
   SetIndexBuffer(0,ind_buffer1);
   SetIndexBuffer(1,ind_buffer2);
   SetIndexBuffer(2,ind_buffer3);
//---- name for DataWindow and indicator subwindow label
   IndicatorShortName("OsMA("+FastEMA+","+SlowEMA+","+SignalSMA+")");
//---- initialization done
   return(0);
  }
```

int IndicatorCounted()

Returns bars count that does not changed after last indicator launch. In most cases same count of index values do not need for recalculation. Used for optimizing calculations.

Sample

```
  int start()
    {
     int limit;
     int counted_bars=IndicatorCounted();
  //---- check for possible errors
     if(counted_bars<0) return(-1);
  //---- last counted bar will be recounted
     if(counted_bars>0) counted_bars--;
     limit=Bars-counted_bars;
  //---- main loop
     for(int i=0; i<limit; i++)
       {
        //---- ma_shift set to 0 because SetIndexShift called abowe
        ExtBlueBuffer[i]=iMA(NULL,0,JawsPeriod,0,MODE_SMMA,PRICE_MEDIAN,i);
        ExtRedBuffer[i]=iMA(NULL,0,TeethPeriod,0,MODE_SMMA,PRICE_MEDIAN,i);
        ExtLimeBuffer[i]=iMA(NULL,0,LipsPeriod,0,MODE_SMMA,PRICE_MEDIAN,i);
       }
  //---- done
     return(0);
    }
```

void IndicatorDigits(int digits)

Sets default precision format for indicators visualization.

Parameters

digits	-	Precision format, number of digits after decimal point.

Sample

```
#property  indicator_separate_window
#property  indicator_buffers 1
#property  indicator_color1  Silver
//---- indicator parameters
extern int FastEMA=12;
extern int SlowEMA=26;
extern int SignalSMA=9;
//---- indicator buffers
double     ind_buffer1[];
double     ind_buffer2[];
double     ind_buffer3[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int init()
  {
//---- 2 additional buffers are used for counting.
```

```
   IndicatorBuffers(3);
//---- drawing settings
   SetIndexStyle(0,DRAW_HISTOGRAM,STYLE_SOLID,3);
   SetIndexDrawBegin(0,SignalSMA);
   IndicatorDigits(MarketInfo(Symbol(),MODE_DIGITS)+2);
//---- 3 indicator buffers mapping
   SetIndexBuffer(0,ind_buffer1);
   SetIndexBuffer(1,ind_buffer2);
   SetIndexBuffer(2,ind_buffer3);
//---- name for DataWindow and indicator subwindow label
   IndicatorShortName("OsMA("+FastEMA+","+SlowEMA+","+SignalSMA+")");
//---- initialization done
   return(0);
  }
```

void IndicatorShortName(string name)

Sets indicator short name for showing on the chart subwindow.

Parameters

name	-	New short name.

Sample

```
#property  indicator_separate_window
#property  indicator_buffers 1
#property  indicator_color1  Silver
//---- indicator parameters
extern int FastEMA=12;
extern int SlowEMA=26;
extern int SignalSMA=9;
//---- indicator buffers
double     ind_buffer1[];
double     ind_buffer2[];
double     ind_buffer3[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int init()
  {
//---- 2 additional buffers are used for counting.
   IndicatorBuffers(3);
//---- drawing settings
   SetIndexStyle(0,DRAW_HISTOGRAM,STYLE_SOLID,3);
   SetIndexDrawBegin(0,SignalSMA);
   IndicatorDigits(MarketInfo(Symbol(),MODE_DIGITS)+2);
//---- 3 indicator buffers mapping
   SetIndexBuffer(0,ind_buffer1);
   SetIndexBuffer(1,ind_buffer2);
   SetIndexBuffer(2,ind_buffer3);
//---- name for DataWindow and indicator subwindow label
   IndicatorShortName("OsMA("+FastEMA+","+SlowEMA+","+SignalSMA+")");
//---- initialization done
   return(0);
  }
```

void SetIndexArrow(int index, int code)

Sets arrow symbol for indicators that draws some lines as arrow.

Parameters

index - Line index. Should be from 0 to 7.
code - Symbol code from Wingdings font or Array constants.

Sample

```
 SetIndexArrow(0, 217);
```

bool SetIndexBuffer(int index, double array[])

Sets buffer for calculating line. The indicated array bound with previously allocated custom indicator buffer. If the function succeeds, the return value is true. If the function fails, the return value is false. To get the detailed error information, call GetLastError().

Parameters

index - Line index. Should be from 0 to 7.
array[] - Array that stores calculated indicator values.

Sample

```
 double ExtBufferSilver[];
 int init()
   {
     SetIndexBuffer(0, ExtBufferSilver); // set buffer for first line
     // ...
   }
```

void SetIndexDrawBegin(int index, int begin)

Sets first bar from what index will be drawn. Index values before draw begin are not significant and does not drawn and not show in the DataWindow.

Parameters

index - Line index. Should be from 0 to 7.
begin - First drawing bar position number.

Sample

```
#property  indicator_separate_window
#property  indicator_buffers 1
#property  indicator_color1  Silver
//---- indicator parameters
extern int FastEMA=12;
extern int SlowEMA=26;
extern int SignalSMA=9;
//---- indicator buffers
double     ind_buffer1[];
double     ind_buffer2[];
double     ind_buffer3[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int init()
  {
//---- 2 additional buffers are used for counting.
   IndicatorBuffers(3);
//---- drawing settings
   SetIndexStyle(0,DRAW_HISTOGRAM,STYLE_SOLID,3);
   SetIndexDrawBegin(0,SignalSMA);
   IndicatorDigits(MarketInfo(Symbol(),MODE_DIGITS)+2);
//---- 3 indicator buffers mapping
   SetIndexBuffer(0,ind_buffer1);
   SetIndexBuffer(1,ind_buffer2);
   SetIndexBuffer(2,ind_buffer3);
//---- name for DataWindow and indicator subwindow label
   IndicatorShortName("OsMA("+FastEMA+","+SlowEMA+","+SignalSMA+")");
//---- initialization done
   return(0);
  }
```

void SetIndexEmptyValue(int index, double value)

Sets drawing line empty value. By default, empty value line is EMPTY_VALUE. Empty values are not drawn and not show in the DataWindow.

Parameters

index	-	Line index. Should be from 0 to 7.
value	-	New empty value.

Sample

```
  SetIndexEmptyValue(6,0.0001);
```

void SetIndexLabel(int index, string text)

Sets drawing line description for showing in the DataWindow.

Parameters

index	-	Line index. Should be from 0 to 7.
text	-	Label text. NULL means that index value does not show in the DataWindow.

Sample

```
//+------------------------------------------------------------------+
//| Ichimoku Kinko Hyo initialization function                       |
//+------------------------------------------------------------------+
int init()
  {
//----
   SetIndexStyle(0,DRAW_LINE);
   SetIndexBuffer(0,Tenkan_Buffer);
   SetIndexDrawBegin(0,Tenkan-1);
   SetIndexLabel(0,"Tenkan Sen");
//----
   SetIndexStyle(1,DRAW_LINE);
   SetIndexBuffer(1,Kijun_Buffer);
   SetIndexDrawBegin(1,Kijun-1);
   SetIndexLabel(1,"Kijun Sen");
//----
   a_begin=Kijun; if(a_begin<Tenkan) a_begin=Tenkan;
   SetIndexStyle(2,DRAW_HISTOGRAM,STYLE_DOT);
   SetIndexBuffer(2,SpanA_Buffer);
   SetIndexDrawBegin(2,Kijun+a_begin-1);
   SetIndexShift(2,Kijun);
//---- Up Kumo bounding line does not show in the DataWindow
   SetIndexLabel(2,NULL);
   SetIndexStyle(5,DRAW_LINE,STYLE_DOT);
   SetIndexBuffer(5,SpanA2_Buffer);
   SetIndexDrawBegin(5,Kijun+a_begin-1);
   SetIndexShift(5,Kijun);
   SetIndexLabel(5,"Senkou Span A");
//----
   SetIndexStyle(3,DRAW_HISTOGRAM,STYLE_DOT);
   SetIndexBuffer(3,SpanB_Buffer);
   SetIndexDrawBegin(3,Kijun+Senkou-1);
   SetIndexShift(3,Kijun);
//---- Down Kumo bounding line does not show in the DataWindow
   SetIndexLabel(3,NULL);
//----
   SetIndexStyle(6,DRAW_LINE,STYLE_DOT);
   SetIndexBuffer(6,SpanB2_Buffer);
   SetIndexDrawBegin(6,Kijun+Senkou-1);
   SetIndexShift(6,Kijun);
   SetIndexLabel(6,"Senkou Span B");
//----
   SetIndexStyle(4,DRAW_LINE);
   SetIndexBuffer(4,Chinkou_Buffer);
   SetIndexShift(4,-Kijun);
   SetIndexLabel(4,"Chinkou Span");
//----
   return(0);
  }
```

void SetIndexShift(int index, int shift)

Sets offset for drawing line. Line will be counted on the current bar, but will be drawn shifted.

Parameters

index	-	Line index. Should be from 0 to 7.
shift	-	Shitf value in bars.

Sample

```
//+------------------------------------------------------------------+
//| Alligator initialization function                                |
//+------------------------------------------------------------------+
int init()
  {
//---- line shifts when drawing
   SetIndexShift(0,JawsShift);
   SetIndexShift(1,TeethShift);
   SetIndexShift(2,LipsShift);
//---- first positions skipped when drawing
   SetIndexDrawBegin(0,JawsShift+JawsPeriod);
   SetIndexDrawBegin(1,TeethShift+TeethPeriod);
   SetIndexDrawBegin(2,LipsShift+LipsPeriod);
//---- 3 indicator buffers mapping
   SetIndexBuffer(0,ExtBlueBuffer);
   SetIndexBuffer(1,ExtRedBuffer);
   SetIndexBuffer(2,ExtLimeBuffer);
//---- drawing settings
   SetIndexStyle(0,DRAW_LINE);
   SetIndexStyle(1,DRAW_LINE);
   SetIndexStyle(2,DRAW_LINE);
//---- index labels
   SetIndexLabel(0,"Gator Jaws");
   SetIndexLabel(1,"Gator Teeth");
   SetIndexLabel(2,"Gator Lips");
//---- initialization done
   return(0);
  }
```

```
void SetIndexStyle(int index, int type, int style=EMPTY, int width=EMPTY, color clr=CLR_NONE)
```

Sets new type, style, width and color for a given indicator line.

Parameters

index	-	Line index. Should be from 0 to 7.
type	-	Shape style.Can be one of Drawing shape style enumeration.
style	-	Drawing style. Except STYLE_SOLID style all other styles valid when width is 1 pixel.Can be one of Shape style enumeration. EMPTY value indicates that style does not changed.
width	-	Line width. valid values - 1,2,3,4,5. EMPTY value indicates that width does not changed.
clr	-	Line color.

Sample

```
 SetIndexStyle(3, DRAW_LINE, EMPTY, 2, Red);
```

```
void SetLevelStyle(int draw_style, int line_width, color clr=CLR_NONE)
```

Function sets new style, width and color of indicator levels.

Parameters

draw_style	-	Drawing style. Except for STYLE_SOLID, all other styles are valid if the width is 1 pixel.Can be one of Shape style constants.EMPTY value indicates that style will not be changed.
line_width	-	Line width. Valid values are 1,2,3,4,5. EMPTY value indicates that width will not be changed.
clr	-	Line color.

Sample

```
//---- show levels as thick red lines
   SetLevelStyle(STYLE_SOLID,2,Red)
```

```
int SetLevelValue(int level, double value)
```

Function sets a new value for the given indicator level.

Parameters

level	-	Level index (0-31).
value	-	Value for the given indicator level.

Sample

```
SetLevelValue(1,3.14);
```

Date & Time functions

datetime CurTime()

Returns the last known server time, number of seconds elapsed from 00:00 January 1, 1970.

Sample

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

int Day()

Returns the current day of the month.

Sample

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

int DayOfWeek()

Returns the current zero based day of the week (0-Sunday,1,2,3,4,5,6).

Sample

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

int DayOfYear()

Returns the current day of the year (1-1 january,..,365(6) - 31 december).

Sample

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

int Hour()

Returns current hour (0,1,2,..23)

Sample

```
bool is_siesta=false;
if(Hour()>=12 || Hour()<17)
    is_siesta=true;
```

datetime LocalTime()

Returns local computer time, number of seconds elapsed from 00:00 January 1, 1970.

Sample

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

int Minute()

Returns current minute (0,1,2,..59).

Sample

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

int Month()

Returns current month as number (1-January,2,3,4,5,6,7,8,9,10,11,12).

Sample

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

int Seconds()

Returns current second (0,1,2,..59).

Sample

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

int TimeDay(datetime date)

Returns day of month (1 - 31) for specified date.

Parameters

date - Datetime is the number of seconds elapsed since midnight (00:00:00), January 1, 1970.

Sample

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

int TimeDayOfWeek(datetime date)

Returns zero based day of week (0-Sunday,1,2,3,4,5,6) for specified date.

Parameters

date - Datetime is the number of seconds elapsed since midnight (00:00:00), January 1, 1970.

Sample

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

int TimeDayOfYear(datetime date)

Returns day (1-1 january,..,365(6) - 31 december) of year for specified date.

Parameters

date - Datetime is the number of seconds elapsed since midnight (00:00:00), January 1, 1970.

Sample

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

int TimeHour(datetime time)

Returns hour for specified time.

Parameters

time - Datetime is the number of seconds elapsed since midnight (00:00:00), January 1, 1970.

Sample

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

int TimeMinute(datetime time)

Returns minute for specified time.

Parameters

time - Datetime is the number of seconds elapsed since midnight (00:00:00), January 1, 1970.

Sample

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

int TimeMonth(datetime time)

Returns month for specified time.

Parameters

time - Datetime is the number of seconds elapsed since midnight (00:00:00), January 1, 1970.

Sample

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

int TimeSeconds(datetime time)

Returns seconds after minute (0 – 59) for specified time.

Parameters

time - Datetime is the number of seconds elapsed since midnight (00:00:00), January 1, 1970.

Sample

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

int TimeYear(datetime time)

Returns year for specified date. Return values can be in range 1970-2037.

Parameters

time - Datetime is the number of seconds elapsed since midnight (00:00:00), January 1, 1970.

Sample

```
int y=TimeYear(CurTime());
```

int Year()

Returns current year.

Sample

```
// return if date before 1 May 2002
if(Year()==2002 && Month()<5)
  return(0);
```

File functions

FileWriteString()

void FileClose(int handle)

Closes file previously opened by FileOpen() functions.

Parameters

handle - File handle, returned by FileOpen() functions

Sample

```
int handle=FileOpen("filename", FILE_CSV|FILE_READ);
if(handle>0)
  {
   // working with file ...
   FileClose(handle);
  }
```

void FileClose(int handle)

Closes file previously opened by FileOpen() functions.

Parameters

handle - File handle, returned by FileOpen() functions

Sample

```
int handle=FileOpen("filename", FILE_CSV|FILE_READ);
if(handle>0)
  {
   // working with file ...
   FileClose(handle);
  }
```

void FileFlush(int handle)

Flushes all data stored in the file buffer to disk.

Parameters

handle - File handle, returned by FileOpen() functions.

Sample

```
int bars_count=Bars;
int handle=FileOpen("mydat.csv",FILE_CSV|FILE_WRITE);
if(handle>0)
  {
   FileWrite(handle, "#","OPEN","CLOSE","HIGH","LOW");
   for(int i=0;i<bars_count;i++)
     FileWrite(handle, i+1,Open[i],Close[i],High[i], Low[i]);
   FileFlush(handle);
   ...
   for(int i=0;i<bars_count;i++)
     FileWrite(handle, i+1,Open[i],Close[i],High[i], Low[i]);
   FileClose(handle);
  }
```

bool FileIsEnding(int handle)

Returns logical true if file pointer is at the end of the file, otherwise returns false. To get the detailed error information, call GetLastError() function.

Parameters

handle - File handle, returned by FileOpen() functions.

Sample

```
if(FileIsEnding(h1))
  {
   FileClose(h1);
   return(false);
  }
```

bool FileIsLineEnding(int handle)

For CSV file returns logical true if file pointer is at the end of the line, otherwise returns false. To get the detailed error information, call GetLastError() function.

Parameters

handle - File handle, returned by FileOpen() function.

Sample

```
if(FileIsLineEnding(h1))
  {
   FileClose(h1);
   return(false);
  }
```

int FileOpen(string filename, int mode, int delimiter=';')

Opens file for input and/or output. Returns a file handle for the opened file. If the function fails, the return value less than 1. To get the detailed error information, call GetLastError() function.
Note: Files can be opened only from terminal_dir\experts\files directory and its subdirectories.

Parameters

filename	-	File name, file may be with any extensions.
mode	-	Open mode. can be one or combination of values: FILE_BIN, FILE_CSV, FILE_READ, FILE_WRITE.
delimiter	-	Delimiter character for csv files. By default passed ';' symbol.

Sample

```
int handle;
handle=FileOpen("my_data.csv",FILE_CSV|FILE_READ,';');
if(handle<1)
  {
   Print("File my_data.dat not found, the last error is ", GetLastError());
   return(false);
  }
```

int FileOpenHistory(string filename, int mode, int delimiter=';')

Opens file in the current history directory for input and/or output. Returns a file handle for the opened file. If the function fails, the return value less than 1. To get the detailed error information, call GetLastError().

Parameters

filename	-	File name, file may be with any extensions.
mode	-	Open mode. can be one or combination of values: FILE_BIN, FILE_CSV, FILE_READ, FILE_WRITE.
delimiter	-	Delimiter character for csv files. By default passed ';' symbol.

Sample

```
int handle=FileOpenHistory("USDX240.HST",FILE_BIN|FILE_WRITE);
if(handle<1)
  {
   Print("Cannot create file USDX240.HST");
   return(false);
  }
```

```
// work with file
// ...
FileClose(handle);
```

int FileReadArray(int handle, object& array[], int start, int count)

Reads the indicated count of elements from the binary file to array. Returns actual read elements count. To get the detailed error information, call GetLastError() function. **Note:** Before reading the data, array must be resized to a sufficient size.

Parameters

handle	-	File handle, returned by FileOpen() function.
array[]	-	Array where data will be stored.
start	-	Storing start position into array.
count	-	Count of elements to read.

Sample

```
int handle;
double varray[10];
handle=FileOpen("filename.dat", FILE_BIN|FILE_READ);
if(handle>0)
  {
   FileReadArray(handle, varray, 0, 10);
   FileClose(handle);
  }
```

int FileReadArray(int handle, object& array[], int start, int count)

Reads the indicated count of elements from the binary file to array. Returns actual read elements count. To get the detailed error information, call GetLastError() function. **Note:** Before reading the data, array must be resized to a sufficient size.

Parameters

handle	-	File handle, returned by FileOpen() function.
array[]	-	Array where data will be stored.
start	-	Storing start position into array.
count	-	Count of elements to read.

Sample

```
int handle;
double varray[10];
handle=FileOpen("filename.dat", FILE_BIN|FILE_READ);
if(handle>0)
  {
   FileReadArray(handle, varray, 0, 10);
   FileClose(handle);
  }
```

int FileReadInteger(int handle, int size=LONG_VALUE)

Read the integer from binary files from the current file position. Integer format size can be 1, 2 or 4 bytes length. If the format size is not specified system attempts to read 4 bytes length value. To get the detailed error information, call GetLastError() function.

Parameters

handle	-	File handle, returned by FileOpen() function.
size	-	Format size. Can be CHAR_VALUE(1 byte), SHORT_VALUE(2 bytes) or LONG_VALUE(4 bytes).

Sample

```
int handle;
int value;
```

```
handle=FileOpen("mydata.dat", FILE_BIN|FILE_READ);
if(handle>0)
  {
   value=FileReadInteger(h1,2);
   FileClose(handle);
  }
```

double FileReadNumber(int handle)

Read the number from the current file position to the delimiter. Only for CSV files. To get the detailed error information, call GetLastError() function.

Parameters

handle	-	File handle, returned by FileOpen() function.

Sample

```
int handle;
int value;
handle=FileOpen("filename.csv", FILE_CSV, ';');
if(handle>0)
  {
   value=FileReadNumber(handle);
   FileClose(handle);
  }
```

string FileReadString(int handle, int length=0)

Read the string from the current file position. Applied to both CSV and binary files. For text files string will be read to the delimiter and for binary file string will be read for the count of characters indicated. To get the detailed error information, call GetLastError() function.

Parameters

handle	-	File handle, returned by FileOpen() function.
length	-	Reading characters count.

Sample

```
int handle;
string str;
handle=FileOpen("filename.csv", FILE_CSV|FILE_READ);
if(handle>0)
  {
   str=FileReadString(handle);
   FileClose(handle);
  }
```

bool FileSeek(int handle, int offset, int origin)

Moves the file pointer to a specified location. The FileSeek() function moves the file pointer associated with handle to a new location that is offset bytes from origin. The next operation on the file occurs at the new location. If successful, function returns TRUE. Otherwise, it returns FALSE. To get the detailed error information, call GetLastError() function.

Parameters

handle	-	File handle, returned by FileOpen() functions.
offset	-	Offset in bytes from origin.
origin	-	Initial position. Value can be one of this constants: SEEK_CUR - from current position, SEEK_SET - from begin, SEEK_END - from end of file.

Sample

```
int handle=FileOpen("filename.csv", FILE_CSV|FILE_READ, ';');
if(handle>0)
```

```
    {
     FileSeek(handle, 10, SEEK_SET);
     FileReadInteger(handle);
     FileClose(handle);
     handle=0;
    }
```

int FileSize(int handle)

Returns file size in bytes. To get the detailed error information, call GetLastError() function.

Parameters

handle	-	File handle, returned by FileOpen() function.

Sample

```
  int handle;
  int size;
  handle=FileOpen("my_table.dat", FILE_BIN|FILE_READ);
  if(handle>0)
    {
     size=FileSize(handle);
     Print("my_table.dat size is ", size, " bytes");
     FileClose(handle);
    }
```

int FileSize(int handle)

Returns file size in bytes. To get the detailed error information, call GetLastError() function.

Parameters

handle - File handle, returned by FileOpen() function.

Sample

```
  int handle;
  int size;
  handle=FileOpen("my_table.dat", FILE_BIN|FILE_READ);
  if(handle>0)
    {
     size=FileSize(handle);
     Print("my_table.dat size is ", size, " bytes");
     FileClose(handle);
    }
```

int FileWrite(int handle, ...)

Writes to the CSV file some values, delimiter inserted automatically. Returns the number of characters written, or a negative value if an error occurs. To get the detailed error information, call GetLastError() function.

Parameters

handle	-	File handle, returned by FileOpen() function.
...	-	User data to write, separated with commas. **Note:** int and double types automatically converted to string,but color, datetime and bool types does not automatically converted and will be writen to file in it's as integers.

Sample

```
  int handle;
  datetime orderOpen=OrderOpenTime();
  handle=FileOpen("filename", FILE_CSV|FILE_WRITE, ';');
  if(handle>0)
    {
```

```
    FileWrite(handle, Close[0], Open[0], High[0], Low[0], TimeToStr(orderOpen));
    FileClose(handle);
   }
```

int FileWriteArray(int handle, object array[], int start, int count)

Writes array to the binary file. Arrays of type int, bool, datetime and color will be written as 4 bytes integers. Arrays of type double will be written as 8 bytes floating point numbers. Arrays of string will be written as one string where elements will be divided by Carriage return and Line feed symbols (0D 0A). Returns the number of elements wrote, or a negative value if an error occurs. To get the detailed error information, call GetLastError() function.

Parameters

handle	-	File handle, returned by FileOpen() function.
array[]	-	Array to write.
start	-	Starting index into array to write.
count	-	Count of elements to write.

Sample

```
int handle;
double BarOpenValues[10];
// copy first ten bars to the array
for(int i=0;i<10; i++)
  BarOpenValues[i]=Open[i];
// writing array to the file
handle=FileOpen("mydata.dat", FILE_BIN|FILE_WRITE);
if(handle>0)
  {
   FileWriteArray(handle, BarOpenValues, 3, 7); // writing last 7 elements
   FileClose(handle);
  }
```

int FileWriteDouble(int handle, double value, int size=DOUBLE_VALUE)

Writes double value to the binary file. If size is FLOAT_VALUE, value will be written as 4 bytes floating point format, else will be written in 8 bytes floating point format. Returns actual written bytes count. To get the detailed error information, call GetLastError() function.

Parameters

handle	-	File handle, returned by FileOpen() function.
value	-	Value to write.
size	-	Optional format flag. It can be any of the following values: DOUBLE_VALUE (8 bytes, default) FLOAT_VALUE (4 bytes).

Sample

```
int handle;
double var1=0.345;
handle=FileOpen("mydata.dat", FILE_BIN|FILE_WRITE);
if(handle<1)
  {
   Print("can't open file error-",GetLastError());
   return(0);
  }
FileWriteDouble(h1, var1, DOUBLE_VALUE);
//...
FileClose(handle);
```

int FileWriteInteger(int handle, int value, int size=LONG_VALUE)

Writes integer value to the binary file. If size is SHORT_VALUE, value will be written as 2 bytes integer, if size is CHAR_VALUE,

value will be written as 1 bytes integer and if size is LONG_VALUE, value will be written as 4 bytes integer. Returns actual written bytes count.
To get the detailed error information, call GetLastError() function.

Parameters

handle	-	File handle, returned by FileOpen() function.
value	-	Value to write.
size	-	Optional format flag. It can be any of the following values: CHAR_VALUE (1 byte), SHORT_VALUE (2 bytes), LONG_VALUE (4 bytes, default).

Sample

```
int handle;
int value=10;
handle=FileOpen("filename.dat", FILE_BIN|FILE_WRITE);
if(handle<1)
  {
   Print("can't open file error-",GetLastError());
   return(0);
  }
FileWriteInteger(handle, value, SHORT_VALUE);
//...
FileClose(handle);
```

int FileWriteString(int handle, string value, int length)

Writes string to the binary file from current file position. Returns actual written bytes count.
To get the detailed error information, call GetLastError() function.

Parameters

handle	-	File handle, returned by FileOpen() function.
value	-	Text to write.
length	-	Counts of characters to write.

Sample

```
int handle;
string str="some string";
handle=FileOpen("filename.bin", FILE_BIN|FILE_WRITE);
  if(handle<1)
  {
   Print("can't open file error-",GetLastError());
   return(0);
  }
FileWriteString(h1, str, 8);
FileClose(handle);
```

Global Variables functions

GlobalVariableCheck()
GlobalVariableDel()
GlobalVariableGet()
GlobalVariableSet()
GlobalVariableSetOnCondition()
GlobalVariablesDeleteAll()

bool GlobalVariableCheck(string name)

Return logical true if global variable exists, otherwise returns false. To get the detailed error information, call GetLastError() function.

Parameters

name - Global variable name.

Sample

```
// check variable before use
if(!GlobalVariableCheck("g1"))
  GlobalVariableSet("g1",1);
```

bool GlobalVariableDel(string name)

Deletes global variable. If the function succeeds, the return value will be true. If the function fails, the return value is false. To get the detailed error information, call GetLastError().

Parameters

name - Global variable name.

Sample

```
// deleting global variable with name "gvar_1"
GlobalVariableDel("gvar_1");
```

double GlobalVariableGet(string name)

Returns global variable value. To check function failure, check error information by calling GetLastError().

Parameters

name - Global variable name.

Sample

```
double v1=GlobalVariableGet("g1");
//---- check function call result
if(GetLastError()!=0) return(false);
//---- continue processing
```

datetime GlobalVariableSet(string name, double value)

Sets global variable value. If it does not exist, the system creates a new variable. If the function succeeds, the return value is last access time. If the function fails, the return value is 0. To get the detailed error information, call GetLastError().

Parameters

name - Global variable name.
value - Numeric value to set.

Sample

```
//---- try to set new value
if(GlobalVariableSet("BarsTotal",Bars)==0)
  return(false);
//---- continue processing
```

bool GlobalVariableSetOnCondition(string name, double value, double check_value)

Sets the new value of the global variable if the current value equals to the third parameter *check_value*. If there is no variable at all, the function will return false and set the value of ERR_GLOBAL_VARIABLE_NOT_FOUND constant to LastError. When successfully executed, the function returns true, otherwise it does false. To receive the information about the error, call GetLastError() function.
The function can be used as a semaphore for the access to common resources.

Parameters

name - Global variable name.
value - Numeric value to set.
check_value - Value to compare with the current global variable value.

Sample

```
int init()
  {
   //---- create global variable
```

```
    GlobalVariableSet("DATAFILE_SEM",0);
    //...
   }

 int start()
   {
    //---- try to lock common resource
    while(!IsStopped())
      {
       //---- locking
       if(GlobalVariableSetOnCondition("DATAFILE_SEM",1,0)==true)  break;
       //---- may be variable deleted?
       if(GetLastError()==ERR_GLOBAL_VARIABLE_NOT_FOUND) return(0);
       //---- sleeping
       Sleep(500);
      }
    //---- resource locked
    // ... do some work
    //---- unlock resource
    GlobalVariableSet("DATAFILE_SEM",0);
   }
```

void GlobalVariablesDeleteAll()

Deletes all global variables. This function never fails.

Sample

```
GlobalVariablesDeleteAll();
```

Math & Trig

- MathAbs()
- MathArccos()
- MathArcsin()
- MathArctan()
- MathCeil()
- MathCos()
- MathExp()
- MathFloor()
- MathLog()
- MathMax()
- MathMin()
- MathMod()
- MathPow()
- MathRand()
- MathRound()
- MathSin()
- MathSqrt()
- MathSrand()
- MathTan()

double MathAbs(double value)

Returns the absolute value (modulus) of the specified numeric value.

Parameters

value	-	Numeric value.

Sample

```
double dx=-3.141593, dy;
// calc MathAbs
dy=MathAbs(dx);
Print("The absolute value of ",dx," is ",dy);
// Output: The absolute value of -3.141593 is 3.141593
```

double MathArccos(double x)

The MathArccos function returns the arccosine of *x* in the range 0 to π radians. If *x* is less than -1 or greater than 1, MathArccos returns an indefinite (same as a quiet NaN).

Parameters

x - Value between -1 and 1 arc cosine of which should be calculated.

Sample

```
double x=0.32696, y;
y=asin(x);
Print("Arcsine of ",x," = ",y);
y=acos(x);
Print("Arccosine of ",x," = ",y);
//Output: Arcsine   of 0.326960=0.333085
//Output: Arccosine of 0.326960=1.237711
```

double MathArcsin(double x)

The MathArcsin function returns the arcsine of *x* in the range -π/2 to π/2 radians. If *x* is less than -1 or greater than 1, arcsine returns an indefinite (same as a quiet NaN).

Parameters

x - Value the arcsine of which should be calculated

Sample

```
double x=0.32696, y;
y=MathArcsin(x);
Print("Arcsine of ",x," = ",y);
y=acos(x);
Print("Arccosine of ",x," = ",y);
//Output: Arcsine   of 0.326960=0.333085
//Output: Arccosine of 0.326960=1.237711
```

double MathArctan(double x)

The MathArctan returns the arctangent of *x*. If *x* is 0, MathArctan returns 0. MathArctan returns a value in the range -π/2 to π/2 radians.

Parameters

x - A number representing a tangent.

Sample

```
double x=-862.42, y;
y=MathArctan(x);
Print("Arctangent of ",x," is ",y);
//Output: Arctangent of -862.42 is -1.5696
```

double MathCeil(double x)

The MathCeil function returns a numeric value representing the smallest integer that is greater than or equal to *x*.

Parameters

x - Numeric value.

Sample

```
double y;
```

```
y=MathCeil(2.8);
Print("The ceil of 2.8 is ",y);
y=MathCeil(-2.8);
Print("The ceil of -2.8 is ",y);
/*Output:
  The ceil of 2.8 is 3
  The ceil of -2.8 is -2*/
```

double MathCos(double value)

Returns the cosine of the specified angle.

Parameters

value - An angle, measured in radians.

Sample

```
double pi=3.1415926535;
double x, y;
x=pi/2;
y=MathSin(x);
Print("MathSin(",x,") = ",y);
y=MathCos(x);
Print("MathCos(",x,") = ",y);
//Output: MathSin(1.5708)=1
//        MathCos(1.5708)=0
```

double MathExp(double d)

Returns value the number **e** raised to the power *d*. On overflow, the function returns INF (infinite) and on underflow, MathExp returns 0.

Parameters

d - A number specifying a power.

Sample

```
double x=2.302585093,y;
y=MathExp(x);
Print("MathExp(",x,") = ",y);
//Output: MathExp(2.3026)=10
```

double MathFloor(double x)

The MathFloor function returns a numeric value representing the largest integer that is less than or equal to *x*.

Parameters

x - Numeric value.

Sample

```
double y;
y=MathFloor(2.8);
Print("The floor of 2.8 is ",y);
y=MathFloor(-2.8);
Print("The floor of -2.8 is ",y);
/*Output:
  The floor of 2.8 is 2
  The floor of -2.8 is -3*/
```

double MathLog(double x)

The MathLog functions return the logarithm of *x* if successful. If *x* is negative, these functions return an indefinite (same as a quiet NaN). If *x* is 0, they return INF (infinite).

Parameters

x - Value whose logarithm is to be found.

Sample

```
double x=9000.0,y;
y=MathLog(x);
Print("MathLog(",x,") = ", y);
//Output: MathLog(9000)=9.10498
```

double MathMax(double value1, double value2)

Returns maximum value of two numeric values.

Parameters

value1	-	First numeric value.
value2	-	Second numeric value.

Sample

```
double result=MathMax(1.08,Bid);
```

double MathMin(double value1, double value2)

Returns minimum value of two numeric values.

Parameters

value1	-	First numeric value.
value2	-	Second numeric value.

Sample

```
double result=MathMin(1.08,Ask);
```

double MathMod(double value, double value2)

Divides two numbers and returns only the remainder.

Parameters

value	-	Dividend value.
value2	-	Divider value.

Sample

```
double x=-10.0,y=3.0,z;
z=MathMod(x,y);
Print("The remainder of ",x," / ",y," is ",z);
//Output: The remainder of -10 / 3 is -1
```

double MathPow(double base, double exponent)

Returns the value of a base expression taken to a specified power.

Parameters

base	-	Base value.
exponent	-	Exponent value.

Sample

```
double x=2.0,y=3.0,z;
z=MathPow(x,y);
Printf(x," to the power of ",y," is ", z);
//Output: 2 to the power of 3 is 8
```

int MathRand()

The MathRand function returns a pseudorandom integer in the range 0 to 0x7fff (32767). Use the MathSrand function to seed the pseudorandom-number generator before calling rand.

Sample

```
MathSrand(LocalTime());
// Display 10 numbers.
for(int i=0;i<10;i++ )
  Print("random value ", MathRand());
```

double MathRound(double value)

Returns value rounded to the nearest integer of the specified numeric value.

Parameters

value - Numeric value to round.

Sample

```
double y=MathRound(2.8);
Print("The round of 2.8 is ",y);
y=MathRound(2.4);
Print("The round of -2.4 is ",y);
//Output: The round of 2.8 is 3
//        The round of -2.4 is -2
```

double MathSin(double value)

Returns the sine of the specified angle.

Parameters

value - An angle, measured in radians.

Sample

```
double pi=3.1415926535;
double x, y;
x=pi/2;
y=MathSin(x);
Print("MathSin(",x,") = ",y);
y=MathCos(x);
Print("MathCos(",x,") = ",y);
//Output: MathSin(1.5708)=1
//        MathCos(1.5708)=0
```

double MathSqrt(double x)

The MathSqrt function returns the square-root of *x*. If *x* is negative, MathSqrt returns an indefinite (same as a quiet NaN).

Parameters

x - Positive numeric value.

Sample

```
double question=45.35, answer;
answer=MathSqrt(question);
if(question<0)
  Print("Error: MathSqrt returns ",answer," answer");
else
  Print("The square root of ",question," is ", answer);
//Output: The square root of 45.35 is 6.73
```

void MathSrand(int seed)

The MathSrand() function sets the starting point for generating a series of pseudorandom integers. To reinitialize the generator, use 1 as the seed argument. Any other value for seed sets the generator to a random starting point. MathRand retrieves the pseudorandom numbers that are generated. Calling MathRand before any call to MathSrand generates the same sequence as calling MathSrand with seed passed as 1.

Parameters

seed - Seed for random-number generation.

Sample

```
MathSrand(LocalTime());
// Display 10 numbers.
for(int i=0;i<10;i++ )
  Print("random value ", MathRand());
```

```
double MathTan(double x)
```

MathTan returns the tangent of *x*. If *x* is greater than or equal to 263, or less than or equal to -263, a loss of significance in the result occurs, in which case the function returns an indefinite (same as a quiet NaN).

Parameters

x - Angle in radians.

Sample

```
double pi=3.1415926535;
double x,y;
x=MathTan(pi/4);
Print("MathTan(",pi/4," = ",x);
//Output: MathTan(0.7856)=1
```

Object functions

ObjectCreate()
ObjectDelete()
ObjectDescription()
ObjectFind()
ObjectGet()
ObjectGetFiboDescription()
ObjectGetShiftByValue()
ObjectGetValueByShift()
ObjectGetVisibility()
ObjectMove()
ObjectName()
ObjectsDeleteAll()
ObjectSet()
ObjectSetFiboDescription()
ObjectSetText()
ObjectSetVisibility()
ObjectsRedraw
ObjectsTotal()
ObjectType()

```
bool              string name, int type, int window, datetime time1, double price1,
ObjectCreate(     datetime time2=0, double price2=0, datetime time3=0, double price3=0)
```

Create object with specified name, type and initial coordinates in the specified window. Count of coordinates related from object type (1-3). If the function succeeds, the return value will be true. If the function succeeds, the return value is true. If the function fails, the return value is false. To get the detailed error information, call GetLastError(). For objects with type OBJ_LABEL first coordinate ignored. To set coordinate for label use ObjectSet() function to set OBJPROP_XDISTANCE and OBJPROP_YDISTANCE properties.
Note: Coordinates must be passed with both part - time and price. For example: Object OBJ_VLINE required 1 coordinate part *time*. But function wants also the seconds part of coordinate *price*.

Parameters

name	-	Unique object name.
type	-	Object type. It can be any of the Object type enumeration values.
window	-	Window index where object will be added. Window index must be greater or equal to 0 and less than WindowsTotal().
time1	-	Time part of first point.
price1	-	Price part of first point.
time2	-	Time part of second point.
price2	-	Price part of second point.
time3	-	Time part of third point.
price3	-	Price part of third point.

Sample

```
// new text object
if(!ObjectCreate("text_object", OBJ_TEXT, 0, D'2004.02.20 12:30', 1.0045))
  {
   Print("error: can't create text_object! code #",GetLastError());
   return(0);
  }
// new label object
if(!ObjectCreate("label_object", OBJ_LABEL, 0, 0, 0))
  {
   Print("error: can't create label_object! code #",GetLastError());
   return(0);
  }
ObjectSet("label_object", OBJPROP_XDISTANCE, 200);
ObjectSet("label_object", OBJPROP_YDISTANCE, 100);
```

bool ObjectDelete(string name)

Deletes object with specified name. If the function succeeds, the return value will be true. If the function succeeds, the return value is true. If the function fails, the return value is false. To get the detailed error information, call GetLastError().

Parameters

name	-	Deleting object name.

Sample

```
ObjectDelete("text_object");
```

string ObjectDescription(string name)

Return object description. To get error information, call GetLastError() function.

Parameters

name	-	Object name.

Sample

```
// writing chart's object list to the file
int    handle, total;
string obj_name,fname;
// file name
fname="objlist_"+Symbol();
handle=FileOpen(fname,FILE_CSV|FILE_WRITE);
if(handle!=false)
  {
   total=ObjectsTotal();
   for(int i=-;i<total;i++)
     {
      obj_name=ObjectName(i);
      FileWrite(handle,"Object "+obj_name+" description=
"+ObjectDescription(obj_name));
     }
```

```
    FileClose(handle);
   }
```

int ObjectFind(string name)

Return object owner's window index. If the function fails, the return value will be -1. To get the detailed error information, call GetLastError() function.

Parameters

name	-	Object name to check.

Sample

```
  if(ObjectFind("line_object2")!=win_idx) return(0);
```

double ObjectGet(string name, int index)

Returns objects property value by index. To check errors, call GetLastError() function.

Parameters

name	-	Object name.
index	-	Object property index. It can be any of the Object properties enumeration values.

Sample

```
  color oldColor=ObjectGet("hline12", OBJPROP_COLOR);
```

string ObjectGetFiboDescription(string name, int index)

Function returns description of Fibonacci level. The amount of Fibonacci levels depends on the object type. The maximum amount of Fibonacci levels never exceeds 32.
To get the detailed error information, call GetLastError() function.

Parameters

name	-	Object name.
index	-	Index of the Fibonacci level.

Sample

```
#include <stdlib.mqh>
  ...
  string text;
  for(int i=0;i<32;i++)
    {
     text=ObjectGetFiboDescription(MyObjectName,i);
     //---- check. may be objects's level count less than 32
     if(GetLastError()!=ERR_NO_ERROR) break;
     Print(MyObjectName,"level: ",i," description: ",text);
    }
```

int ObjectGetShiftByValue(string name, double value)

Calculates and returns bar index for the indicated price. Calculated by first and second coordinate. Applied to trendlines. To get the detailed error information, call GetLastError() function.

Parameters

name	-	Object name.
value	-	Price value.

Sample

```
  int shift=ObjectGetShiftByValue("MyTrendLine#123", 1.34);
```

double ObjectGetValueByShift(string name, int shift)

Calculates and returns price value for the indicated bar. Calculated by first and second coordinate. Applied to trendlines. To get the detailed error information, call GetLastError() function.

Parameters

name	-	Object name
shift	-	Bar index.

Sample

```
double price=ObjectGetValueByShift("MyTrendLine#123", 11);
```

int ObjectGetVisibility(string name)

Function returns flags of the object visibility on the chart. Value can be single or combined (bitwise addition) of object visibility constants.

Parameters

name	-	Object name.

Sample

```
  // is object visible on the chart?
  if((ObjectGetVisibility()&OBJ_PERIOD_M5)!=0 && Period()==PERIOD_M5)
    {
     // working with object
    }
```

bool ObjectMove(string name, int point, datetime time1, double price1)

Moves objects point on the chart. Objects can have from one to three points related to its type. If the function succeeds, the return value will be true. If the function fails, the return value will be false. To get the detailed error information, call GetLastError().

Parameters

name	-	Object name.
point	-	Coordinate index.
time1	-	New time value.
price1	-	New price value.

Sample

```
ObjectMove("MyTrend", 1, D'2005.02.25 12:30', 1.2345);
```

string ObjectName(int index)

Returns object name by index.

Parameters

index	-	Object index on the chart. Object index must be greater or equal to 0 and less than ObjectsTotal().

Sample

```
int    obj_total=ObjectsTotal();
string name;
for(int i=0;i<obj_total;i++)
  {
   name=ObjectName(i);
   Print(i,"Object name is " + name);
  }
```

int ObjectsDeleteAll(int window, int type=EMPTY)

Removes all objects with specified type and on the specified subwindow of the chart. Returns removed objects count.

Parameters

window	-	Window index from objects will be deleted. Window index must be greater or equal to 0 and less than WindowsTotal().
type	-	Optional object type to delete.It can be any of the Object type enumeration values or EMPTY constant to delete all objects with any types.

Sample

```
  ObjectsDeleteAll(2, OBJ_HLINE); // removes all horizontal line objects from window
3 (index 2).
```

bool ObjectSet(string name, int index, double value)

Changes named objects property with new value. If the function succeeds, the return value will be true. If the function fails, the return value will be false. To get the detailed error information, call GetLastError().

Parameters

name	-	Object name.
index	-	Object value index. It can be any of Object properties enumeration values.
value	-	New value for property.

Sample

```
  // moving first coord to last bar time
  ObjectSet("MyTrend", OBJPROP_TIME1, Time[0]);
  // setting second fibo level
  ObjectSet("MyFibo", OBJPROP_FIRSTLEVEL+1, 1.234);
  // setting object visibility. object will be shown only on 15 minute and 1 hour
charts
  ObjectSet("MyObject", OBJPROP_TIMEFRAMES, OBJ_PERIOD_M15 | OBJ_PERIOD_H1);
```

bool ObjectSetFiboDescription(string name, int index, string text)

Function assigns a new description to a Fibonacci level. The amount of Fibonacci levels depends on the object type. The maximum amount of Fibonacci levels never exceeds 32.
To get the detailed error information, call GetLastError() function.

Parameters

name	-	Object name.
index	-	Index of the Fibonacci level (0-31).
text	-	New description to be assigned to the Fibonacci level.

Sample

```
  ObjectSetFiboDescription("MyFiboObject,2,"Second line");
```

bool ObjectSetText(string name, string text, int font_size, string font=NULL, color text_color=CLR_NONE)

Sets object description. If the function succeeds, the return value will be true. If the function fails, the return value will be false. To get the detailed error information, call GetLastError() function.

Parameters

name	-	Object name.
text	-	Some text.
font_size	-	Font size in points.
font	-	Font name.
text_color	-	Text color.

Sample

```
  ObjectSetText("text_object", "Hello world!", 10, "Times New Roman", Green);
```

int ObjectSetVisibility(string name, int flag)

Function sets new value to the object visibility property. Function returns previous value.

Parameters

name	-	Object name.
flag	-	New value of the object visibility property. Value can be single or combined (bitwise addition) of object visibility constants.

Sample

```
  // The object will be shown on 1-hour charts only.
```

```
ObjectSetVisibility("MyObj1",OBJ_PERIOD_H1);
```

void ObjectsRedraw()

Redraws all objects on the char.

Sample

```
ObjectsRedraw();
```

int ObjectsTotal()

Returns total count of objects on the chart.

Sample

```
int    obj_total=ObjectsTotal();
string name;
for(int i=0;i<obj_total;i++)
  {
   name=ObjectName(i);
   Print(i,"Object name is for object #",i," is " + name);
  }
```

int ObjectType(string name)

Returns Object type enumeration value.

Parameters

name - Object name.

Sample

```
if(ObjectType("line_object2")!=OBJ_HLINE) return(0);
```

Pre-defined Variables

- Ask
- Bars
- Bid
- Close
- Digits
- High
- Low
- Open
- Point
- Time
- Volume

double Ask

Ask price (the Buyer's price).

```
if(iRSI(NULL,0,14,PRICE_CLOSE,0)<25)
  {
   OrderSend(Symbol(),OP_BUY,Lots,Ask,3,Ask-StopLoss*Point,Ask+TakeProfit*Point,
             "My order #2",3,D'2005.10.10 12:30',Red);
   return;
  }
```

int Bars

Number of bars on the chart.

```
int counter=1;
for(int i=1;i<=Bars;i++)
  {
   Print(Close[i-1]);
  }
```

double Bid

Bid price (the Seller's price).

```
if(iRSI(NULL,0,14,PRICE_CLOSE,0)>75)
  {
   OrderSend("EURUSD",OP_SELL,Lots,Bid,3,Bid+StopLoss*Point,Bid-TakeProfit*Point,
             "My order #2",3,D'2005.10.10 12:30',Red);
   return(0);
  }
```

double Close[]

Returns the closing price of the bar being referenced.

```
int handle, bars=Bars;
handle=FileOpen("file.csv",FILE_CSV|FILE_WRITE,';');
if(handle>0)
  {
   // write table columns headers
   FileWrite(handle, "Time;Open;High;Low;Close;Volume");
   // write data
   for(int i=0; i<bars; i++)
     FileWrite(handle, Time[i], Open[i], High[i], Low[i], Close[i], Volume[i]);
   FileClose(handle);
  }
```

int Digits

Number of digits after decimal point for the current symbol.

```
Print(DoubleToStr(Close[i-1], Digits));
```

double High[]

Returns the highest price of the bar referenced.

```
int handle, bars=Bars;
handle=FileOpen("file.csv", FILE_CSV|FILE_WRITE, ';');
if(handle>0)
  {
   // write table columns headers
   FileWrite(handle, "Time;Open;High;Low;Close;Volume");
   // write data
   for(int i=0; i<bars; i++)
     FileWrite(handle, Time[i], Open[i], High[i], Low[i], Close[i], Volume[i]);
   FileClose(handle);
  }
```

double Low[]

Returns the lowest price of the bar referenced.

```
int handle, bars=Bars;
handle=FileOpen("file.csv", FILE_CSV|FILE_WRITE, ";");
if(handle>0)
  {
   // write table columns headers
   FileWrite(handle, "Time;Open;High;Low;Close;Volume");
   // write data
   for(int i=0; i<bars; i++)
     FileWrite(handle, Time[i], Open[i], High[i], Low[i], Close[i], Volume[i]);
   FileClose(handle);
  }
```

double Open[]

Returns the opening price of the bar referenced.

```
int handle, bars=Bars;
handle=FileOpen("file.csv", FILE_CSV|FILE_WRITE, ';');
if(handle>0)
  {
   // write table columns headers
   FileWrite(handle, "Time;Open;High;Low;Close;Volume");
   // write data
   for(int i=0; i<bars; i++)
     FileWrite(handle, Time[i], Open[i], High[i], Low[i], Close[i], Volume[i]);
   FileClose(handle);
  }
```

double Point

Point value for the current chart.

```
OrderSend(Symbol(),OP_BUY,Lots,Ask,3,0,Ask+TakeProfit*Point,Red);
```

datetime Time[]

Open time of the bars. Datetime is the number of seconds elapsed from 00:00 January 1, 1970.

```
int handle, bars=Bars;
handle=FileOpen("file.csv", FILE_CSV|FILE_WRITE, ';');
if(handle>0)
  {
   // write table columns headers
   FileWrite(handle, "Time;Open;High;Low;Close;Volume");
   // write data
   for(int i=0; i<bars; i++)
     FileWrite(handle, Time[i], Open[i], High[i], Low[i], Close[i], Volume[i]);
   FileClose(handle);
  }
```

double Volume[]

Returns the ticks count for the referenced bar.

```
int handle, bars=Bars;
handle=FileOpen("file.csv", FILE_CSV|FILE_WRITE, ';');
if(handle>0)
  {
   // write table columns headers
   FileWrite(handle, "Time;Open;High;Low;Close;Volume");
   // erite data
   for(int i=0; i<bars; i++)
     FileWrite(handle, Time[i], Open[i], High[i], Low[i], Close[i], Volume[i]);
   FileClose(handle);
  }
```

Standard Constants

Applied price enumeration

Applied price constants. It can be any of the following values:

Constant	Value	Description
PRICE_CLOSE	0	Close price.
PRICE_OPEN	1	Open price.
PRICE_HIGH	2	High price.
PRICE_LOW	3	Low price.
PRICE_MEDIAN	4	Median price, (high+low)/2.
PRICE_TYPICAL	5	Typical price, (high+low+close)/3.
PRICE_WEIGHTED	6	Weighted close price, (high+low+close+close)/4.

Drawing shape style enumeration

Drawing shape style enumeration for SetIndexStyle() function.
It can be any of the following values:

Constant	Value	Description
DRAW_LINE	0	Drawing line.
DRAW_SECTION	1	Drawing sections.
DRAW_HISTOGRAM	2	Drawing histogram.
DRAW_ARROW	3	Drawing arrows (symbols).
DRAW_NONE	12	No drawing.

Drawing style. Valid when width=1. It can be any of the following values:

Constant	Value	Description
STYLE_SOLID	0	The pen is solid.
STYLE_DASH	1	The pen is dashed.
STYLE_DOT	2	The pen is dotted.
STYLE_DASHDOT	3	The pen has alternating dashes and dots.
STYLE_DASHDOTDOT	4	The pen has alternating dashes and double dots.

Error codes

The GetLastError() function return codes. Error code constants defined at **stderror.mqh** file. To print text messages use *ErrorDescription()* function defined at **stdlib.mqh** file.

```
#include <stdlib.mqh>
void SendMyMessage(string text)
  {
   int check;
   SendMail("some subject", text);
   check=GetLastError();
   if(check!=ERR_NO_MQLERROR) Print("Cannot send message, error:
",ErrorDescription(check));
  }
```

Error codes returned from trade server.

Constant	Value	Description
ERR_NO_ERROR	0	No error returned.
ERR_NO_RESULT	1	No error returned, but the result is unknown.
ERR_COMMON_ERROR	2	Common error.
ERR_INVALID_TRADE_PARAMETERS	3	Invalid trade parameters.
ERR_SERVER_BUSY	4	Trade server is busy.
ERR_OLD_VERSION	5	Old version of the client terminal.
ERR_NO_CONNECTION	6	No connection with trade server.
ERR_NOT_ENOUGH_RIGHTS	7	Not enough rights.
ERR_TOO_FREQUENT_REQUESTS	8	Too frequent requests.
ERR_MALFUNCTIONAL_TRADE	9	Malfunctional trade operation.
ERR_ACCOUNT_DISABLED	64	Account disabled.
ERR_INVALID_ACCOUNT	65	Invalid account.
ERR_TRADE_TIMEOUT	128	Trade timeout.
ERR_INVALID_PRICE	129	Invalid price.
ERR_INVALID_STOPS	130	Invalid stops.
ERR_INVALID_TRADE_VOLUME	131	Invalid trade volume.
ERR_MARKET_CLOSED	132	Market is closed.
ERR_TRADE_DISABLED	133	Trade is disabled.
ERR_NOT_ENOUGH_MONEY	134	Not enough money.
ERR_PRICE_CHANGED	135	Price changed.
ERR_OFF_QUOTES	136	Off quotes.
ERR_BROKER_BUSY	137	Broker is busy.
ERR_REQUOTE	138	Requote.
ERR_ORDER_LOCKED	139	Order is locked.
ERR_LONG_POSITIONS_ONLY_ALLOWED	140	Long positions only allowed.
ERR_TOO_MANY_REQUESTS	141	Too many requests.
ERR_TRADE_MODIFY_DENIED	145	Modification denied because order too close to market.
ERR_TRADE_CONTEXT_BUSY	146	Trade context is busy.

MQL4 run time error codes

Constant	Value	Description
ERR_NO_MQLERROR	4000	No error.
ERR_WRONG_FUNCTION_POINTER	4001	Wrong function pointer.
ERR_ARRAY_INDEX_OUT_OF_RANGE	4002	Array index is out of range.
ERR_NO_MEMORY_FOR_FUNCTION_CALL_STACK	4003	No memory for function call stack.
ERR_RECURSIVE_STACK_OVERFLOW	4004	Recursive stack overflow.
ERR_NOT_ENOUGH_STACK_FOR_PARAMETER	4005	Not enough stack for parameter.
ERR_NO_MEMORY_FOR_PARAMETER_STRING	4006	No memory for parameter string.
ERR_NO_MEMORY_FOR_TEMP_STRING	4007	No memory for temp string.
ERR_NOT_INITIALIZED_STRING	4008	Not initialized string.
ERR_NOT_INITIALIZED_ARRAYSTRING	4009	Not initialized string in array.
ERR_NO_MEMORY_FOR_ARRAYSTRING	4010	No memory for array string.
ERR_TOO_LONG_STRING	4011	Too long string.
ERR_REMAINDER_FROM_ZERO_DIVIDE	4012	Remainder from zero divide.
ERR_ZERO_DIVIDE	4013	Zero divide.
ERR_UNKNOWN_COMMAND	4014	Unknown command.
ERR_WRONG_JUMP	4015	Wrong jump (never generated error).
ERR_NOT_INITIALIZED_ARRAY	4016	Not initialized array.
ERR_DLL_CALLS_NOT_ALLOWED	4017	DLL calls are not allowed.
ERR_CANNOT_LOAD_LIBRARY	4018	Cannot load library.
ERR_CANNOT_CALL_FUNCTION	4019	Cannot call function.
ERR_EXTERNAL_EXPERT_CALLS_NOT_ALLOWED	4020	Expert function calls are not allowed.
ERR_NOT_ENOUGH_MEMORY_FOR_RETURNED_STRING	4021	Not enough memory for temp string returned from function.
ERR_SYSTEM_BUSY	4022	System is busy (never generated error).
ERR_INVALID_FUNCTION_PARAMETERS_COUNT	4050	Invalid function parameters count.
ERR_INVALID_FUNCTION_PARAMETER_VALUE	4051	Invalid function parameter value.
ERR_STRING_FUNCTION_INTERNAL_ERROR	4052	String function internal error.
ERR_SOME_ARRAY_ERROR	4053	Some array error.
ERR_INCORRECT_SERIES_ARRAY_USING	4054	Incorrect series array using.
ERR_CUSTOM_INDICATOR_ERROR	4055	Custom indicator error.
ERR_INCOMPATIBLE_ARRAYS	4056	Arrays are incompatible.
ERR_GLOBAL_VARIABLES_PROCESSING_ERROR	4057	Global variables processing error.
ERR_GLOBAL_VARIABLE_NOT_FOUND	4058	Global variable not found.
ERR_FUNCTION_NOT_ALLOWED_IN_TESTING_MODE	4059	Function is not allowed in testing mode.
ERR_FUNCTION_NOT_CONFIRMED	4060	Function is not confirmed.
ERR_SEND_MAIL_ERROR	4061	Send mail error.
ERR_STRING_PARAMETER_EXPECTED	4062	String parameter expected.
ERR_INTEGER_PARAMETER_EXPECTED	4063	Integer parameter expected.
ERR_DOUBLE_PARAMETER_EXPECTED	4064	Double parameter expected.
ERR_ARRAY_AS_PARAMETER_EXPECTED	4065	Array as parameter expected.
ERR_HISTORY_WILL_UPDATED	4066	Requested history data in updating state.
ERR_END_OF_FILE	4099	End of file.
ERR_SOME_FILE_ERROR	4100	Some file error.
ERR_WRONG_FILE_NAME	4101	Wrong file name.
ERR_TOO_MANY_OPENED_FILES	4102	Too many opened files.
ERR_CANNOT_OPEN_FILE	4103	Cannot open file.

Ichimoku Kinko Hyo modes enumeration

Ichimoku Kinko Hyo source of data. Used in iChimoku() indicators.
It can be one of the following values:

Constant	Value	Description
MODE_TENKANSEN	1	Tenkan-sen.
MODE_KIJUNSEN	2	Kijun-sen.
MODE_SENKOUSPANA	3	Senkou Span A.
MODE_SENKOUSPANB	4	Senkou Span B.
MODE_CHINKOUSPAN	5	Chinkou Span.

Indicators line identifiers

Indicator line identifiers used in iMACD(), iRVI() and iStochastic() indicators.
It can be one of the following values:

Constant	Value	Description
MODE_MAIN	0	Base indicator line.
MODE_SIGNAL	1	Signal line.

Indicator line identifiers used in iADX() indicator.

Constant	Value	Description
MODE_MAIN	0	Base indicator line.
MODE_PLUSDI	1	+DI indicator line.
MODE_MINUSDI	2	-DI indicator line.

Indicator line identifiers used in iBands(), iEnvelopes(), iEnvelopesOnArray(), iFractals() and iGator() indicators.

Constant	Value	Description
MODE_UPPER	1	Upper line.
MODE_LOWER	2	Lower line.

Market information identifiers

Market information identifiers, used with MarketInfo() function.
It can be any of the following values:

Constant	Value	Description
MODE_LOW	1	Low day price.
MODE_HIGH	2	High day price.
MODE_TIME	5	The last incoming quotation time.
MODE_BID	9	Last incoming bid price.
MODE_ASK	10	Last incoming ask price.
MODE_POINT	11	Point size.
MODE_DIGITS	12	Digits after decimal point.
MODE_SPREAD	13	Spread value in points.
MODE_STOPLEVEL	14	Stop level in points.
MODE_LOTSIZE	15	Lot size in the base currency.
MODE_TICKVALUE	16	Tick value.
MODE_TICKSIZE	17	Tick size.
MODE_SWAPLONG	18	Swap of the long position.
MODE_SWAPSHORT	19	Swap of the short position.
MODE_STARTING	20	Market starting date (usually used for future markets).
MODE_EXPIRATION	21	Market expiration date (usually used for future markets).
MODE_TRADEALLOWED	22	Trade is allowed for the symbol.
MODE_MINLOT	22	The minimum lot size in points.
MODE_LOTSTEP	22	Step for changing lots in points.

MessageBox return codes

The MessageBox() function return codes.
If a message box has a **Cancel** button, the function returns the IDCANCEL value if either the ESC key is pressed or the **Cancel** button is selected. If the message box has no **Cancel** button, pressing ESC has no effect.
Note: MessageBox return codes defined in the WinUser32.mqh file

Constant	Value	Description
IDOK	1	**OK** button was selected.
IDCANCEL	2	**Cancel** button was selected.
IDABORT	3	**Abort** button was selected.
IDRETRY	4	**Retry** button was selected.
IDIGNORE	5	**Ignore** button was selected.
IDYES	6	**Yes** button was selected.
IDNO	7	**No** button was selected.
IDTRYAGAIN	10	**Try Again** button was selected.
IDCONTINUE	11	**Continue** button was selected.

MessageBox behavior flags

The MessageBox function flags specify the contents and behavior of the dialog box. This value can be a combination of flags from the following groups of flags.
Note: MessageBox return codes defined in the WinUser32.mqh file

To indicate the buttons displayed in the message box, specify one of the following values.

Constant	Value	Description
MB_OK	0x00000000	The message box contains one push button: OK. This is the default.
MB_OKCANCEL	0x00000001	The message box contains two push buttons: OK and Cancel.
MB_ABORTRETRYIGNORE	0x00000002	The message box contains three push buttons: Abort, Retry, and Ignore.
MB_YESNOCANCEL	0x00000003	The message box contains three push buttons: Yes, No, and Cancel.
MB_YESNO	0x00000004	The message box contains two push buttons: Yes and No.
MB_RETRYCANCEL	0x00000005	The message box contains two push buttons: Retry and Cancel.
MB_CANCELTRYCONTINUE	0x00000006	Windows 2000: The message box contains three push buttons: Cancel, Try Again, Continue. Use this message box type instead of MB_ABORTRETRYIGNORE.

To display an icon in the message box, specify one of the following values.

Constant	Value	Description
MB_ICONSTOP, MB_ICONERROR, MB_ICONHAND	0x00000010	A stop-sign icon appears in the message box.
MB_ICONQUESTION	0x00000020	A question-mark icon appears in the message box.
MB_ICONEXCLAMATION, MB_ICONWARNING	0x00000030	An exclamation-point icon appears in the message box.
MB_ICONINFORMATION, MB_ICONASTERISK	0x00000040	An icon consisting of a lowercase letter i in a circle appears in the message box.

To indicate the default button, specify one of the following values.

Constant	Value	Description
MB_DEFBUTTON1	0x00000000	The first button is the default button. MB_DEFBUTTON1 is the default unless MB_DEFBUTTON2, MB_DEFBUTTON3, or MB_DEFBUTTON4 is specified.
MB_DEFBUTTON2	0x00000100	The second button is the default button.
MB_DEFBUTTON3	0x00000200	The third button is the default button.
MB_DEFBUTTON4	0x00000300	The fourth button is the default button.

Moving Average method enumeration

Moving Average calculation method. used with iAlligator(), iEnvelopes(), iEnvelopesOnArray, iForce(), iGator(), iMA(), iMAOnArray(), iStdDev(), iStdDevOnArray(), iStochastic() indicators.
It can be any of the following values:

Constant	Value	Description
MODE_SMA	0	Simple moving average,
MODE_EMA	1	Exponential moving average,
MODE_SMMA	2	Smoothed moving average,
MODE_LWMA	3	Linear weighted moving average.

Object properties enumeration

Object value index used with ObjectGet() and ObjectSet() functions. It can be any of the following values:

Constant	Value	Description
OBJPROP_TIME1	0	Datetime value to set/get first coordinate time part.
OBJPROP_PRICE1	1	Double value to set/get first coordinate price part.
OBJPROP_TIME2	2	Datetime value to set/get second coordinate time part.
OBJPROP_PRICE2	3	Double value to set/get second coordinate price part.
OBJPROP_TIME3	4	Datetime value to set/get third coordinate time part.
OBJPROP_PRICE3	5	Double value to set/get third coordinate price part.
OBJPROP_COLOR	6	Color value to set/get object color.
OBJPROP_STYLE	7	Value is one of STYLE_SOLID, STYLE_DASH, STYLE_DOT, STYLE_DASHDOT, STYLE_DASHDOTDOT constants to set/get object line style.
OBJPROP_WIDTH	8	Integer value to set/get object line width. Can be from 1 to 5.
OBJPROP_BACK	9	Boolean value to set/get background drawing flag for object.
OBJPROP_RAY	10	Boolean value to set/get ray flag of object.
OBJPROP_ELLIPSE	11	Boolean value to set/get ellipse flag for fibo arcs.
OBJPROP_SCALE	12	Double value to set/get scale object property.
OBJPROP_ANGLE	13	Double value to set/get angle object property in degrees.
OBJPROP_ARROWCODE	14	Integer value or arrow enumeration to set/get arrow code object property.
OBJPROP_TIMEFRAMES	15	Value can be one or combination (bitwise addition) of object visibility constants to set/get timeframe object property.
OBJPROP_DEVIATION	16	Double value to set/get deviation property for Standard deviation objects.
OBJPROP_FONTSIZE	100	Integer value to set/get font size for text objects.
OBJPROP_CORNER	101	Integer value to set/get anchor corner property for label objects. Must be from 0-3.
OBJPROP_XDISTANCE	102	Integer value to set/get anchor X distance object property in pixels.
OBJPROP_YDISTANCE	103	Integer value is to set/get anchor Y distance object property in pixels.
OBJPROP_FIBOLEVELS	200	Integer value to set/get Fibonacci object level count. Can be from 0 to 32.
OBJPROP_FIRSTLEVEL+*n*	210	Fibonacci object level index, where *n* is level index to set/get. Can be from 0 to 31.

Object type enumeration

Object type identifier constants used with ObjectCreate(), ObjectsDeleteAll() and ObjectType() functions. It can be any of the following values:
Objects can have 1-3 coordinates related to type.

Constant	Value	Description
OBJ_VLINE	0	Vertical line. Uses time part of first coordinate.
OBJ_HLINE	1	Horizontal line. Uses price part of first coordinate.
OBJ_TREND	2	Trend line. Uses 2 coordinates.
OBJ_TRENDBYANGLE	3	Trend by angle. Uses 1 coordinate. To set angle of line use ObjectSet() function.
OBJ_REGRESSION	4	Regression. Uses time parts of first two coordinates.
OBJ_CHANNEL	5	Channel. Uses 3 coordinates.
OBJ_STDDEVCHANNEL	6	Standard deviation channel. Uses time parts of first two coordinates.
OBJ_GANNLINE	7	Gann line. Uses 2 coordinate, but price part of second coordinate ignored.
OBJ_GANNFAN	8	Gann fan. Uses 2 coordinate, but price part of second coordinate ignored.
OBJ_GANNGRID	9	Gann grid. Uses 2 coordinate, but price part of second coordinate ignored.
OBJ_FIBO	10	Fibonacci retracement. Uses 2 coordinates.
OBJ_FIBOTIMES	11	Fibonacci time zones. Uses 2 coordinates.
OBJ_FIBOFAN	12	Fibonacci fan. Uses 2 coordinates.
OBJ_FIBOARC	13	Fibonacci arcs. Uses 2 coordinates.
OBJ_EXPANSION	14	Fibonacci expansions. Uses 3 coordinates.
OBJ_FIBOCHANNEL	15	Fibonacci channel. Uses 3 coordinates.
OBJ_RECTANGLE	16	Rectangle. Uses 2 coordinates.
OBJ_TRIANGLE	17	Triangle. Uses 3 coordinates.
OBJ_ELLIPSE	18	Ellipse. Uses 2 coordinates.
OBJ_PITCHFORK	19	Andrews pitchfork. Uses 3 coordinates.
OBJ_CYCLES	20	Cycles. Uses 2 coordinates.
OBJ_TEXT	21	Text. Uses 1 coordinate.
OBJ_ARROW	22	Arrows. Uses 1 coordinate.
OBJ_LABEL	23	Text label. Uses 1 coordinate in pixels.

Object visibility enumeration

Time frames where object may be shown. Used in ObjectSet() function to set OBJPROP_TIMEFRAMES property.

Constant	Value	Description
OBJ_PERIOD_M1	0x0001	Object shown is only on 1-minute charts.
OBJ_PERIOD_M5	0x0002	Object shown is only on 5-minute charts.
OBJ_PERIOD_M15	0x0004	Object shown is only on 15-minute charts.
OBJ_PERIOD_M30	0x0008	Object shown is only on 30-minute charts.
OBJ_PERIOD_H1	0x0010	Object shown is only on 1-hour charts.
OBJ_PERIOD_H4	0x0020	Object shown is only on 4-hour charts.
OBJ_PERIOD_D1	0x0040	Object shown is only on daily charts.
OBJ_PERIOD_W1	0x0080	Object shown is only on weekly charts.
OBJ_PERIOD_MN1	0x0100	Object shown is only on monthly charts.
OBJ_ALL_PERIODS	0x01FF	Object shown is on all timeframes.
NULL	0	Object shown is on all timeframes.
EMPTY	-1	Hidden object on all timeframes.

Predefined Arrow codes enumeration

Predefined Arrow codes enumeration. Arrows code constants. It can be one of the following values:

Constant	Value	Description
SYMBOL_THUMBSUP	67	Thumb up symbol (👍).
SYMBOL_THUMBSDOWN	68	Thumb down symbol (👎).
SYMBOL_ARROWUP	241	Arrow up symbol (⇧).
SYMBOL_ARROWDOWN	242	Arrow down symbol (⇩).
SYMBOL_STOPSIGN	251	Stop sign symbol (✖).
SYMBOL_CHECKSIGN	252	Check sign symbol (✓).

Special Arrow codes that exactly points to price and time. It can be one of the following values:

Constant	Value	Description
	1	Upwards arrow with tip rightwards (↱).
	2	Downwards arrow with tip rightwards (↳).
	3	Left pointing triangle (◀).
	4	En Dash symbol (–).
SYMBOL_LEFTPRICE	5	Left sided price label.
SYMBOL_RIGHTPRICE	6	Right sided price label.

Series array identifier

Series array identifier used with ArrayCopySeries(), Highest() and Lowest() functions. It can be any of the following values:

Constant	Value	Description
MODE_OPEN	0	Open price.
MODE_LOW	1	Low price.
MODE_HIGH	2	High price.
MODE_CLOSE	3	Close price.
MODE_VOLUME	4	Volume, used in Lowest() and Highest() functions.
MODE_TIME	5	Bar open time, used in ArrayCopySeries() function.

Special constants

Special constants used to indicate parameters and variables states. It can be one of the following values:

Constant	value	Description
NULL	0	Indicates empty state of the string.
EMPTY	-1	Indicates empty state of the parameter.
EMPTY_VALUE	0x7FFFFFFF	Default custom indicator empty value.
CLR_NONE	0xFFFFFFFF	Indicates empty state of colors.
WHOLE_ARRAY	0	Used with array functions. Indicates that all array elements will be processed.

Time frame enumeration

Time frame on the chart. It can be any of the following values:

Constant	Value	Description
PERIOD_M1	1	1 minute.
PERIOD_M5	5	5 minutes.
PERIOD_M15	15	15 minutes.
PERIOD_M30	30	30 minutes.
PERIOD_H1	60	1 hour.
PERIOD_H4	240	4 hour.
PERIOD_D1	1440	Daily.
PERIOD_W1	10080	Weekly.
PERIOD_MN1	43200	Monthly.
0 (zero)	0	Time frame used on the chart.

Trade operation enumeration

Operation type for the OrderSend() function. It can be any of the following values:

Constant	Value	Description
OP_BUY	0	Buying position.
OP_SELL	1	Selling position.
OP_BUYLIMIT	2	Buy limit pending position.
OP_SELLLIMIT	3	Sell limit pending position.
OP_BUYSTOP	4	Buy stop pending position.
OP_SELLSTOP	5	Sell stop pending position.

Uninitialize reason codes

Uninitialize reason codes returned by UninitializeReason() function. It can be any one of the following values:

Constant	Value	Description
REASON_REMOVE	1	Expert removed from chart.
REASON_RECOMPILE	2	Expert recompiled.
REASON_CHARTCHANGE	3	symbol or timeframe changed on the chart.
REASON_CHARTCLOSE	4	Chart closed.
REASON_PARAMETERS	5	Inputs parameters was changed by user.
REASON_ACCOUNT	6	Other account activated.

Wingdings symbols

Wingdings font symbols used with Arrow objects.

32	✏ 33	✂ 34	✁ 35	👓 36	🕭 37	🕮 38	🕯 39	☎ 40	✆ 41	🖂 42	🖃 43	📪 44	📫 45	📬 46	📭 47
📁 48	📂 49	📄 50	🗏 51	🗐 52	🗄 53	⌛ 54	🖮 55	🖰 56	🖲 57	🖳 58	🖴 59	🖫 60	🖬 61	✇ 62	✍ 63
🖎 64	✌ 65	👌 66	👍 67	👎 68	☜ 69	☞ 70	☝ 71	☟ 72	🖐 73	☺ 74	😐 75	☹ 76	💣 77	☠ 78	🏳 79
🏱 80	✈ 81	☼ 82	💧 83	❄ 84	🕆 85	✞ 86	🕈 87	✠ 88	✡ 89	☪ 90	☯ 91	ॐ 92	☸ 93	♈ 94	♉ 95
♊ 96	♋ 97	♌ 98	♍ 99	♎ 100	♏ 101	♐ 102	♑ 103	♒ 104	♓ 105	🙰 106	🙵 107	● 108	🔾 109	■ 110	□ 111
🞐 112	❑ 113	❒ 114	⬧ 115	⧫ 116	◆ 117	❖ 118	⬥ 119	⌧ 120	⮹ 121	⌘ 122	🏵 123	🏶 124	🙶 125	🙷 126	▯ 127
⓪ 128	① 129	② 130	③ 131	④ 132	… 133	⑥ 134	⑦ 135	⑧ 136	‰ 137	⑩ 138	⓿ 139	❶ 140	❷ 141	❸ 142	❹ 143
❺ 144	‘ 145	’ 146	“ 147	” 148	❿ 149	– 150	— 151	🙡 152	🙣 153	🙞 154	🙜 155	🙝 156	🙟 157	· 158	• 159
▪ 160	○ 161	⭕ 162	🞅 163	🞆 164	◉ 165	🔿 166	▪ 167	◻ 168	🟂 169	✦ 170	★ 171	✶ 172	📫 173	✹ 174	✵ 175
⯐ 176	⌖ 177	⟡ 178	⌑ 179	⯑ 180	✪ 181	✰ 182	🕐 183	🕑 184	🕒 185	🕓 186	🕔 187	🕕 188	🕖 189	🕗 190	🕘 191
🕙 192	🕚 193	🕛 194	⮰ 195	⮲ 196	⮱ 197	⮳ 198	⮴ 199	⮶ 200	⮵ 201	⮷ 202	🙐 203	🙒 204	🙑 205	🙓 206	🙔 207
🙖 208	🙕 209	🙗 210	🙘 211	🙙 212	⌫ 213	⌦ 214	⮘ 215	⮚ 216	⮙ 217	⮛ 218	⮈ 219	⮊ 220	⮉ 221	⮋ 222	🡨 223
🡪 224	🡩 225	🡫 226	🡬 227	🡭 228	🡯 229	🡮 230	🡸 231	🡺 232	🡹 233	🡻 234	🡼 235	🡽 236	🡿 237	🡾 238	⇦ 239
⇨ 240	⇧ 241	⇩ 242	⬄ 243	⇳ 244	⬁ 245	⬀ 246	⬃ 247	⬂ 248	▭ 249	▫ 250	✗ 251	✓ 252	☒ 253	☑ 254	🪟 255

Web colors table

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

String functions

- **StringConcatenate()**
- **StringFind()**
- **StringGetChar()**
- **StringLen()**
- **StringSetChar()**
- **StringSubstr()**
- **StringTrimLeft()**
- **StringTrimRight()**

```
string StringConcatenate(... )
```

Writes data to the string and returns it. Parameters can be of any type. Arrays cannot be passed to the StringConcatenate() function. Arrays should be printed elementwise. Data of double type printed with 4 decimal digits after point. To print with more precision use DoubleToStr() function. Data of bool, datetime and color types will be printed as its numeric presentation. To print values of datetime type as string convert it by TimeToStr() function. StringConcatenate() function work faster than concatenating strings by + operator.
See also: Print(), Alert() and Comment() functions.

Parameters

... - Any values, separated by commas.

Sample

```
  string text;
  text=StringConcatenate("Account free margin is ", AccountFreeMargin(), "Current
time is ", TimeToStr(CurTime()));
// slow text="Account free margin is " + AccountFreeMargin() + "Current time is " +
TimeToStr(CurTime())
  Print(text);
```

int StringFind(string text, string matched_text, int start=0)

Scans this string for the first match of a substring.

Parameters

text	-	String to search for.
matched_text	-	String to search for.
start	-	Starting index in the string.

Sample

```
  string text="The quick brown dog jumps over the lazy fox";
  int index=StringFind(text, "dog jumps", 0);
  if(index!=16)
    Print("oops!");
```

int StringGetChar(string text, int pos)

Returns character (code) from specified position in the string.

Parameters

text	-	String where character will be retrieved.
pos	-	Char zero based position in the string.

Sample

```
  int char_code=StringGetChar("abcdefgh", 3);
  // char code 'c' is 99
```

int StringLen(string text)

Returns character count of a string.

Parameters

text - String to calculate length.

Sample

```
  string str="some text";
  if(StringLen(str)<5) return(0);
```

string StringSetChar(string text, int pos, int value)

Returns string copy with changed character at the indicated position with new value.

Parameters

text	-	String where character will be changed.
pos	-	Zero based character position in the string. Can be from 0 to StringLen()-1.
value	-	New char ASCII code.

Sample

```
  string str="abcdefgh";
  string str1=StringSetChar(str, 3, 'D');
  // str1 is "abcDefgh"
```

string StringSubstr(string text, int start, int count=EMPTY)

Extracts a substring from text string, starting at position (zero-based).
The function returns a copy of the extracted substring if possible, otherwise returns empty string.

Parameters

text	-	String from substring will be extracted.
start	-	Substring starting index
count	-	Character count.

Sample

```
string text="The quick brown dog jumps over the lazy fox";
string substr=StringSubstr(text, 4, 5);
// subtracted string is "quick" word
```

string StringTrimLeft(string text)

Call the function to trim leading white space characters from the string. StringTrimLeft removes new line, space, and tab characters. The function returns a copy of the trimmed string if possible, otherwise returns empty string. Returns new string with changes.

Parameters

text	-	String to trim left.

Sample

```
string str1="  Hello world   ";
string str2=StringTrimLeft(str);
// after trimming the str2 variable will be "Hello World   "
```

string StringTrimRight(string text)

Call the function to trim leading white space characters from the string. StringTrimRight removes new line, space, and tab characters. The function returns a copy of the trimmed string if possible, otherwise return empty string.

Parameters

text	-	String to trim right.

Sample

```
string str1="  Hello world   ";
string str2=StringTrimRight(str);
// after trimming the str2 variable will be "  Hello World"
```

Technical Indicator calls

```
double iAC(string symbol, int timeframe, int shift)
```

Calculates the Bill Williams' Accelerator/Decelerator oscillator and returns its value.

Parameters

symbol	-	Symbol the data of which should be used to calculate indicator. NULL means the current symbol.
timeframe	-	Time frame. It can be any of Time frame enumeration values.
shift	-	Shift relative to the current bar (number of periods back), where the data should be taken from.

Sample

```
double result=iAC(NULL, 0, 1);
```

```
double iAD(string symbol, int timeframe, int shift)
```

Calculates the Accumulation/Distribution indicator and returns its value.

Parameters

symbol	-	Symbol the data of which should be used to calculate indicator. NULL means the current symbol.
timeframe	-	Time frame. It can be any of Time frame enumeration values.
shift	-	Shift relative to the current bar (number of periods back), where the data should be taken from.

Sample

```
double result=iAD(NULL, 0, 1);
```

```
double        string symbol, int timeframe, int jaw_period, int jaw_shift,
iAlligator(   int teeth_period, int teeth_shift, int lips_period, int lips_shift,
              int ma_method, int applied_price, int mode, int shift)
```

Calculates the Bill Williams' Alligator and returns its value.

Parameters

symbol	-	Symbol the data of which should be used to calculate indicator. NULL means the current symbol.
timeframe	-	Time frame. It can be any of Time frame enumeration values.
jaw_period	-	Jaw period.
jaw_shift	-	Jaw line shift.
teeth_period	-	Teeth period.
teeth_shift	-	Teeth line shift.
lips_period	-	Lips period.
lips_shift	-	Lips line shift.
ma_method	-	MA method. It can be any of Moving Average method enumeration value.
applied_price	-	Applied price. It can be any of Applied price enumeration values.
mode	-	Source of data. It can be any of the following values: MODE_GATORJAW - Gator Jaw (blue) balance line, MODE_GATORTEETH - Gator Teeth (red) balance line, MODE_GATORLIPS - Gator Lips (green) balance line.
shift	-	Shift relative to the current bar (number of periods back), where the data should be taken from.

Sample

```
  double jaw_val=iAlligator(NULl, 0, 13, 8, 8, 5, 5, 3, MODE_SMMA, PRICE_MEDIAN,
MODE_GATORJAW, 1);
```

double iADX(string symbol, int timeframe, int period, int applied_price, int mode, int shift)

Calculates the Movement directional index and returns its value.

Parameters

symbol	-	Symbol the data of which should be used to calculate indicator. NULL means the current symbol.
timeframe	-	Time frame. It can be any of Time frame enumeration values.
period	-	Number of periods for calculation.
applied_price	-	Applied price. It can be any of Applied price enumeration values.
mode	-	Indicator line array index. It can be any of the Indicators line identifiers enumeration value.
shift	-	Shift relative to the current bar (number of periods back), where the data should be taken from.

Sample

```
  if(iADX(NULL,0,14,PRICE_HIGH,MODE_MAIN,0)>iADX(NULL,0,14,PRICE_HIGH,MODE_PLUSDI,0))
return(0);
```

double iATR(string symbol, int timeframe, int period, int shift)

Calculates the Indicator of the average true range and returns its value.

Parameters

symbol	-	Symbol the data of which should be used to calculate indicator. NULL means the current symbol.
timeframe	-	Time frame. It can be any of Time frame enumeration values.
period	-	Number of periods for calculation.
shift	-	Shift relative to the current bar (number of periods back), where the data should be taken from.

Sample

```
  if(iATR(NULL,0,12,0)>iATR(NULL,0,20,0)) return(0);
```

double iAO(string symbol, int timeframe, int shift)

Calculates the Bill Williams' Awesome oscillator and returns its value.

Parameters

symbol	-	Symbol the data of which should be used to calculate indicator. NULL means the current symbol.
timeframe	-	Time frame. It can be any of Time frame enumeration values.
shift	-	Shift relative to the current bar (number of periods back), where the data should be taken from.

Sample

```
  double val=iAO(NULL, 0, 2);
```

```
double          string symbol, int timeframe, int period, int applied_price,
iBearsPower(    int shift)
```

Calculates the Bears Power indicator and returns its value.

Parameters

symbol	-	Symbol the data of which should be used to calculate indicator. NULL means the current symbol.
timeframe	-	Time frame. It can be any of Time frame enumeration values.
period	-	Number of periods for calculation.
applied_price	-	Applied price. It can be any of Applied price enumeration values.
shift	-	Shift relative to the current bar (number of periods back), where the data should be taken from.

Sample

```
double val=iBearsPower(NULL, 0, 13,PRICE_CLOSE,0);
```

```
double     string symbol, int timeframe, int period, int deviation, int bands_shift,
iBands(    int applied_price, int mode, int shift)
```

Calculates the Bollinger bands indicator and returns its value.

Parameters

symbol	-	Symbol the data of which should be used to calculate indicator. NULL means the current symbol.
timeframe	-	Time frame. It can be any of Time frame enumeration values.
period	-	Number of periods for calculation.
deviation	-	Deviation.
bands_shift	-	Bands shift.
applied_price	-	Applied price. It can be any of Applied price enumeration values.
mode	-	Indicator line array index. It can be any of the Indicators line identifiers enumeration value.
shift	-	Shift relative to the current bar (number of periods back), where the data should be taken from.

Sample

```
if(iBands(NULL,0,20,2,0,PRICE_LOW,MODE_LOWER,0)>Low[0]) return(0);
```

```
double          double array[], int total, int period, double deviation,
iBandsOnArray(  int bands_shift, int mode, int shift)
```

Calculates the Bollinger bands indicator and returns its value.

Parameters

array[]	-	Array with data.
total	-	The number of items to be counted.
period	-	Number of periods for calculation.
deviation	-	Deviation.
bands_shift	-	Bands shift.
mode	-	Series array identifier. It can be any of the Series array identifier enumeration values.
shift	-	Shift relative to the current bar (number of periods back), where the data should be taken from.

Sample

```
if(iBands(ExtBuffer,total,2,0,MODE_LOWER,0)>Low[0]) return(0);
```

```
double          string symbol, int timeframe, int period, int applied_price,
iBullsPower(    int shift)
```

Calculates the Bulls Power indicator and returns its value.

Parameters

symbol	-	Symbol the data of which should be used to calculate indicator. NULL means the current symbol.
timeframe	-	Time frame. It can be any of Time frame enumeration values.
period	-	Number of periods for calculation.
applied_price	-	Applied price. It can be any of Applied price enumeration values.
shift	-	Shift relative to the current bar (number of periods back), where the data should be taken from.

Sample

```
double val=iBullsPower(NULL, 0, 13,PRICE_CLOSE,0);
double iCCI(string symbol, int timeframe, int period, int applied_price, int shift)
```

Calculates the Commodity channel index and returns its value.

Parameters

symbol	-	Symbol the data of which should be used to calculate indicator. NULL means the current symbol.
timeframe	-	Time frame. It can be any of Time frame enumeration values.
period	-	Number of periods for calculation.
applied_price	-	Applied price. It can be any of Applied price enumeration values.
shift	-	Shift relative to the current bar (number of periods back), where the data should be taken from.

Sample

```
if(iCCI(NULL,0,12,0)>iCCI(NULL,0,20,0)) return(0);
```

```
double iCCIOnArray(double array[], int total, int period, int shift)
```

Calculates the Commodity channel index and returns its value.

Parameters

array[]	-	Array with data.
total	-	The number of items to be counted.
period	-	Number of periods for calculation.
shift	-	Shift relative to the current bar (number of periods back), where the data should be taken from.

Sample

```
if(iCCIOnArray(ExtBuffer,total,12,0)>iCCI(NULL,0,20,PRICE_OPEN, 0)) return(0);
```

```
double iCustom(string symbol, int timeframe, string name, ... , int mode, int shift)
```

Calculates the Custom indicator and returns its value.

Parameters

symbol	-	Symbol the data of which should be used to calculate indicator. NULL means current symbol.
timeframe	-	Time frame. It can be any of Time frame enumeration values.
name	-	Custom indicator compiled program name.
...	-	Parameters set (if needed).
mode	-	Line index. Can be from 0 to 7.
shift	-	Shift relative to the current bar (number of periods back), where the data should be taken from.

Sample

```
double val=iCustom(NULL, 0, "SampleInd",13,1,0);
```

```
double iDeMarker(string symbol, int timeframe, int period, int shift)
```

Calculates the DeMarker indicator and returns its value.

Parameters

symbol	-	Symbol the data of which should be used to calculate indicator. NULL means the current symbol.
timeframe	-	Time frame. It can be any of Time frame enumeration values.
period	-	Number of periods for calculation.
shift	-	Shift relative to the current bar (number of periods back), where the data should be taken from.

Sample

```
double val=iDeMarker(NULL, 0, 13, 1);
```

```
double      string symbol, int timeframe, int ma_period, int ma_method,
iEnvelopes( int ma_shift, int applied_price, double deviation, int mode, int shift)
```

Calculates the Envelopes indicator and returns its value.

Parameters

symbol	-	Symbol the data of which should be used to calculate indicator. NULL means the current symbol.
timeframe	-	Time frame. It can be any of Time frame enumeration values.
ma_period	-	Number of periods for calculation.
ma_method	-	MA method. It can be any of Moving Average method enumeration value.
ma_shift	-	MA shift. Indicator line offset relate to the chart by timeframe.
applied_price	-	Applied price. It can be any of Applied price enumeration values.
deviation	-	Deviation.
mode	-	Indicator line array index. It can be any of Indicators line identifiers enumeration value.
shift	-	Shift relative to the current bar (number of periods back), where the data should be taken from.

Sample

```
double val=iEnvelopes(NULL, 0, 13,MODE_SMA,10,PRICE_CLOSE,0.2,MODE_UPPER,0);
```

```
double                  double array[], int total, int ma_period, int ma_method,
iEnvelopesOnArray(      int ma_shift, double deviation, int mode, int shift)
```

Calculates the Envelopes indicator counted on buffer and returns its value.

Parameters

array[]	-	Array with data.
total	-	The number of items to be counted.
ma_period	-	Number of periods for calculation.
ma_method	-	MA method. It can be any of Moving Average method enumeration value.
ma_shift	-	MA shift. Indicator line offset relate to the chart by timeframe.
deviation	-	Deviation.
mode	-	Indicator line array index. It can be any of Indicators line identifiers enumeration value.
shift	-	Shift relative to the current bar (number of periods back), where the data should be taken from.

Sample

```
double val=iEnvelopesOnArray(ExtBuffer, 0, 13, MODE_SMA, 0.2, MODE_UPPER,0 );
```

```
double       string symbol, int timeframe, int period, int ma_method,
iForce(      int applied_price, int shift)
```

Calculates the Force index and returns its value.

Parameters

symbol	-	Symbol the data of which should be used to calculate indicator. NULL means current symbol.
timeframe	-	Time frame. It can be any of Time frame enumeration values.
period	-	Number of periods for calculation.
ma_method	-	MA method. It can be any of Moving Average method enumeration value.
applied_price	-	Applied price. It can be any of Applied price enumeration values.
shift	-	Shift relative to the current bar (number of periods back), where the data should be taken from.

Sample

```
double val=iForce(NULL, 0, 13,MODE_SMA,PRICE_CLOSE,0);
```

```
double iFractals(string symbol, int timeframe, int mode, int shift)
```

Calculates the Fractals and returns its value.

Parameters

symbol	-	Symbol the data of which should be used to calculate indicator. NULL means the current symbol.
timeframe	-	Time frame. It can be any of Time frame enumeration values.
mode	-	Indicator line array index. It can be any of the Indicators line identifiers enumeration value.
shift	-	Shift relative to the current bar (number of periods back), where the data should be taken from.

Sample

```
double val=iFractals(NULL, 0, MODE_UPPER,0);
```

```
double    string symbol, int timeframe, int jaw_period, int jaw_shift,
iGator(   int teeth_period, int teeth_shift, int lips_period, int lips_shift,
          int ma_method, int applied_price, int mode, int shift)
```

Calculates the Gator Oscillator and returns its value.

Parameters

symbol	-	Symbol the data of which should be used to calculate indicator. NULL means the current symbol.
timeframe	-	Time frame. It can be any of Time frame enumeration values.
jaw_period	-	Jaw period.
jaw_shift	-	Jaw line shift.
teeth_period	-	Teeth period.
teeth_shift	-	Teeth line shift.
lips_period	-	Lips period.
lips_shift	-	Lips line shift.
ma_method	-	MA method. It can be any of Moving Average method enumeration value.
applied_price	-	Applied price. It can be any of Applied price enumeration values.
mode	-	Indicator line array index. It can be any of Indicators line identifiers enumeration value.
shift	-	Shift relative to the current bar (number of periods back), where the data should be taken from.

Sample

```
  double jaw_val=iGator(NULL, 0, 13, 8, 8, 5, 5, 3, MODE_SMMA, PRICE_MEDIAN,
MODE_UPPER, 1);
```

```
double          string symbol, int timeframe, int tenkan_sen, int kijun_sen,
iIchimoku(      int senkou_span_b, int mode, int shift)
```

Calculates the Ichimoku Kinko Hyo and returns its value.

Parameters

symbol	-	Symbol the data of which should be used to calculate indicator. NULL means the current symbol.
timeframe	-	Time frame. It can be any of Time frame enumeration values.
tenkan_sen	-	Tenkan Sen.
kijun_sen	-	Kijun Sen.
senkou_span_b	-	Senkou SpanB.
mode	-	Source of data. It can be one of the Ichimoku Kinko Hyo mode enumeration.
shift	-	Shift relative to the current bar (number of periods back), where the data should be taken from.

Sample

```
  double tenkan_sen=iIchimoku(NULL, 0, 9, 26, 52, MODE_TENKANSEN, 1);
```

```
double iBWMFI(string symbol, int timeframe, int shift)
```

Calculates the Bill Williams Market Facilitation index and returns its value.

Parameters

symbol	-	Symbol the data of which should be used to calculate indicator. NULL means the current symbol.
timeframe	-	Time frame. It can be any of Time frame enumeration values.
shift	-	Shift relative to the current bar (number of periods back), where the data should be taken from.

Sample

```
  double val=iBWMFI(NULL, 0, 0);
```

```
double           string symbol, int timeframe, int period, int applied_price,
iMomentum(       int shift)
```

Calculates the Momentum indicator and returns its value.

Parameters

symbol	-	Symbol the data of which should be used to calculate indicator.NULL means the current symbol.
timeframe	-	Time frame. It can be any of Time frame enumeration values.
period	-	Number of periods for calculation.
applied_price	-	Applied price. It can be any of Applied price enumeration values.
shift	-	Shift relative to the current bar (number of periods back), where the data should be taken from.

Sample

```
  if(iMomentum(NULL,0,12,PRICE_CLOSE,0)>iMomentum(NULL,0,20,PRICE_CLOSE,0))
```

```
return(0);
```

double iMomentumOnArray(double array[], int total, int period, int shift)

Calculates the Momentum indicator counted on buffer and returns its value.

Parameters

array[]	-	Array with data.
total	-	The number of items to be counted.
period	-	Number of periods for calculation.
shift	-	Shift relative to the current bar (number of periods back), where the data should be taken from.

Sample

```
  if(iMomentumOnArray(mybuffer,100,12,0)>iMomentumOnArray(mubuffer,100,20,0))
return(0);
```

double iMFI(string symbol, int timeframe, int period, int shift)

Calculates the Money flow index and returns its value.

Parameters

symbol	-	Symbol the data of which should be used to calculate indicator. NULL means the current symbol.
timeframe	-	Time frame. It can be any of Time frame enumeration values.
period	-	Number of periods for calculation.
shift	-	Shift relative to the current bar (number of periods back), where the data should be taken from.

Sample

```
  if(iMFI(NULL,0,14,0)>iMFI(NULL,0,14,1)) return(0);
```

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

Calculates the Moving average indicator and returns its value.

Parameters

symbol	-	Symbol the data of which should be used to calculate indicator. NULL means the current symbol.
timeframe	-	Time frame. It can be any of Time frame enumeration values.
period	-	Number of periods for calculation.
ma_shift	-	MA shift. Indicators line offset relate to the chart by timeframe.
ma_method	-	MA method. It can be any of the Moving Average method enumeration value.
applied_price	-	Applied price. It can be any of Applied price enumeration values.
shift	-	Shift relative to the current bar (number of periods back), where the data should be taken from.

Sample

```
  AlligatorJawsBuffer[i]=iMA(NULL,0,13,8,MODE_SMMA,PRICE_MEDIAN,i);
```

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

Calculates the Moving average counted on buffer and returns its value.

Parameters

array[]	-	Array with data.
total	-	The number of items to be counted. 0 means whole array.
period	-	Number of periods for calculation.
ma_shift	-	MA shift
ma_method	-	MA method. It can be any of the Moving Average method enumeration value.
shift	-	Shift relative to the current bar (number of periods back), where the data should be taken from.

Sample

```
   double macurrent=iMAOnArray(ExtBuffer,0,5,0,MODE_LWMA,0);
   double macurrentslow=iMAOnArray(ExtBuffer,0,10,0,MODE_LWMA,0);
   double maprev=iMAOnArray(ExtBuffer,0,5,0,MODE_LWMA,1);
   double maprevslow=iMAOnArray(ExtBuffer,0,10,0,MODE_LWMA,1);
   //----
   if(maprev<maprevslow && macurrent>=macurrentslow)
     Alert("crossing up");
```

double iOsMA(string symbol, int timeframe, int fast_ema_period, int slow_ema_period, int signal_period, int applied_price, int shift)

Calculates the Moving Average of Oscillator and returns its value.

Parameters

symbol	-	Symbol the data of which should be used to calculate indicator. NULL means the current symbol.
timeframe	-	Time frame. It can be any of Time frame enumeration values.
fast_ema_period	-	Number of periods for fast moving average calculation.
slow_ema_period	-	Nmber of periods for slow moving average calculation.
signal_period	-	Number of periods for signal moving average calculation.
applied_price	-	Applied price. It can be any of Applied price enumeration values.
shift	-	Shift relative to the current bar (number of periods back), where the data should be taken from.

Sample

```
  if(iOsMA(NULL,0,12,26,9,PRICE_OPEN,1)>iOsMA(NULL,0,12,26,9,PRICE_OPEN,0))
return(0);
```

double iMACD(string symbol, int timeframe, int fast_ema_period, int slow_ema_period, int signal_period, int applied_price, int mode, int shift)

Calculates the Moving averages convergence/divergence and returns its value.

Parameters

symbol	-	Symbol the data of which should be used to calculate indicator. NULL means the current symbol.
timeframe	-	Time frame. It can be any of Time frame enumeration values.
fast_ema_period	-	Number of periods for fast moving average calculation.
slow_ema_period	-	Number of periods for slow moving average calculation.
signal_period	-	Number of periods for signal moving average calculation.
applied_price	-	Applied price. It can be any of Applied price enumeration values.
mode	-	Indicator line array index. It can be any of the Indicators line identifiers enumeration value.
shift	-	Shift relative to the current bar (number of periods back), where the data should be taken from.

Sample

```
  if(iMACD(NULL,0,12,26,9,PRICE_CLOSE,MODE_MAIN,0)>iMACD(NULL,0,12,26,9,PRICE_CLOSE,M
ODE_SIGNAL,0)) return(0);
```

double iOBV(string symbol, int timeframe, int applied_price, int shift)

Calculates the On Balance Volume indicator and returns its value.

Parameters

symbol	-	Symbol the data of which should be used to calculate indicator. NULL means the current symbol.
timeframe	-	Time frame. It can be any of Time frame enumeration values.
applied_price	-	Applied price. It can be any of Applied price enumeration values.
shift	-	Shift relative to the current bar (number of periods back), where the data should be taken from.

Sample

```
  double val=iOBV(NULL, 0, PRICE_CLOSE, 1);
```

double iSAR(string symbol, int timeframe, double step, double maximum, int shift)

Calculates the Parabolic Sell and Reverse system and returns its value.

Parameters

symbol	-	Symbol the data of which should be used to calculate indicator. NULL means the current symbol.
timeframe	-	Time frame. It can be any of Time frame enumeration values.
step	-	Increment, usually 0.02.
maximum	-	Maximum value, usually 0.2.
shift	-	Shift relative to the current bar (number of periods back), where the data should be taken from.

Sample

```
if(iSAR(NULL,0,0.02,0.2,0)>Close[0]) return(0);
```

double iRSI(string symbol, void timeframe, int period, int applied_price, int shift)

Calculates the Relative strength index and returns its value.

Parameters

symbol	-	Symbol the data of which should be used to calculate indicator. NULL means the current symbol.
timeframe	-	Time frame. It can be any of Time frame enumeration values.
period	-	Number of periods for calculation.
applied_price	-	Applied price. It can be any of Applied price enumeration values.
shift	-	Shift relative to the current bar (number of periods back), where the data should be taken from.

Sample

```
if(iRSI(NULL,0,14,PRICE_CLOSE,0)>iRSI(NULL,0,14,PRICE_CLOSE,1)) return(0);
```

double iRSIOnArray(double array[], int total, int period, int shift)

Calculates the Relative strength index counted on buffer and returns its value.

Parameters

array[]	-	Array with data.
total	-	The number of items to be counted.
period	-	Number of periods for calculation.
shift	-	Shift relative to the current bar (number of periods back), where the data should be taken from.

Sample

```
if(iRSIOnBuffer(ExtBuffer,1000,14,0)>iRSI(NULL,0,14,PRICE_CLOSE,1)) return(0);
```

double iRVI(string symbol, int timeframe, int period, int mode, int shift)

Calculates the Relative Vigor index and returns its value.

Parameters

symbol	-	Symbol the data of which should be used to calculate indicator. NULL means the current symbol.
timeframe	-	Time frame. It can be any of Time frame enumeration values.
period	-	Number of periods for calculation.
mode	-	Indicator line array index. It can be any of Indicators line identifiers enumeration value.
shift	-	Shift relative to the current bar (number of periods back), where the data should be taken from.

Sample

```
double val=iRVI(NULL, 0, 10,MODE_MAIN,0);
```

double iStdDev(string symbol, int timeframe, int ma_period, int ma_method, int ma_shift, int applied_price, int shift)

Calculates the Standard Deviation indicator and returns its value.

Parameters

symbol	-	Symbol the data of which should be used to calculate indicator. NULL means the current symbol.
timeframe	-	Time frame. It can be any of Time frame enumeration values.
ma_period	-	MA period.
ma_method	-	MA method. It can be any of Moving Average method enumeration value.
ma_shift	-	MA shift.
applied_price	-	Applied price. It can be any of Applied price enumeration values.
shift	-	Shift relative to the current bar (number of periods back), where the data should be taken from.

Sample

```
 double val=iStdDev(NULL,0,10,MODE_EMA,0,PRICE_CLOSE,0);
```

```
double              double array[], int total, int ma_period, int ma_method,
iStdDevOnArray(     int ma_shift, int shift)
```

Calculates the Standard Deviation counted on buffer and returns its value.

Parameters

array[]	-	Array with data.
total	-	The number of items to be counted.
ma_period	-	MA period.
ma_method	-	MA method. It can be any of Moving Average method enumeration value.
ma_shift	-	iMA shift.
shift	-	Shift relative to the current bar (number of periods back), where the data should be taken from.

Sample

```
 double val=iStdDevOnArray(ExtBuffer,100,10,MODE_EMA,0,0);
```

```
double          string symbol, int timeframe, int %Kperiod, int %Dperiod, int slowing,
iStochastic(    int method, int price_field, int mode, int shift)
```

Calculates the Stochastic oscillator and returns its value.

Parameters

symbol	-	Symbol the data of which should be used to calculate indicator. NULL means the current symbol.
timeframe	-	Time frame. It can be any of Time frame enumeration values.
%Kperiod	-	%K line period.
%Dperiod	-	%D line period.
slowing	-	Slowing value.
method	-	MA method. It can be any ofMoving Average method enumeration value.
price_field	-	Price field parameter. Can be one of this values: 0 - Low/High or 1 - Close/Close.
mode	-	Indicator line array index. It can be any of the Indicators line identifiers enumeration value.
shift	-	Shift relative to the current bar (number of periods back), where the data should be taken from.

Sample

```
 if(iStochastic(NULL,0,5,3,3,MODE_SMA,0,MODE_MAIN,0)>iStochastic(NULL,0,5,3,3,MODE_S
MA,0,MODE_SIGNAL,0))
   return(0);
```

```
double iWPR(string symbol, int timeframe, int period, int shift)
```

Calculates the Larry William's percent range indicator and returns its value.

Parameters

symbol	-	Symbol the data of which should be used to calculate indicator. NULL means the current symbol.
timeframe	-	Time frame. It can be any of Time frame enumeration values.
period	-	Number of periods for calculation.
shift	-	Shift relative to the current bar (number of periods back), where the data should be taken from.

Sample

```
 if(iWPR(NULL,0,14,0)>iWPR(NULL,0,14,1)) return(0);
```

int iBars(string symbol, int timeframe)

Returns the number of bars on the specified chart.

Parameters

symbol	-	Symbol the data of which should be used to calculate indicator. NULL means the current symbol.
timeframe	-	Time frame. It can be any of Time frame enumeration values.

Sample

```
  Print("Bar count on the 'EUROUSD' symbol with PERIOD_H1
is",iBars("EUROUSD",PERIOD_H1));
```

int iBarShift(string symbol, int timeframe, datetime time, bool exact=false)

Search for bar by open time. The function returns bar shift with the open time specified. If the bar having the specified open time is absent the function will return, depending on the *exact* parameter, -1 or the nearest bar shift.

Parameters

symbol	-	Symbol the data of which should be used to calculate indicator. NULL means the current symbol.
timeframe	-	Time frame. It can be any of Time frame enumeration values.
time	-	value to find (bar's open time).
exact	-	Return mode when bar not found. false - iBarShift returns nearest. true - iBarShift returns -1.

Sample

```
  datetime some_time=D'2004.03.21 12:00';
  int      shift=iBarShift("EUROUSD",PERIOD_M1,some_time);
  Print("shift of bar with open time ",TimeToStr(some_time)," is ",shift);
```

double iClose(string symbol, int timeframe, int shift)

Returns **Close** value for the bar of indicated *symbol* with *timeframe* and *shift*. If local history is empty (not loaded), function returns 0.

Parameters

symbol	-	Symbol the data of which should be used to calculate indicator. NULL means the current symbol.
timeframe	-	Time frame. It can be any of Time frame enumeration values.
shift	-	Shift relative to the current bar (number of periods back), where the data should be taken from.

Sample

```
  Print("Current bar for USDCHF H1: ",iTime("USDCHF",PERIOD_H1,i),", ",
iOpen("USDCHF",PERIOD_H1,i),", ",
                                      iHigh("USDCHF",PERIOD_H1,i),", ",
iLow("USDCHF",PERIOD_H1,i),", ",
                                      iClose("USDCHF",PERIOD_H1,i),", ",
iVolume("USDCHF",PERIOD_H1,i));
```

double iHigh(string symbol, int timeframe, int shift)

Returns **High** value for the bar of indicated *symbol* with *timeframe* and *shift*. If local history is empty (not loaded), function returns 0.

Parameters

symbol	-	Symbol on that data need to calculate indicator. NULL means current symbol.
timeframe	-	Time frame. It can be any of Time frame enumeration values.
shift	-	Shift relative to the current bar (number of periods back), where the data should be taken from.

Sample

```
  Print("Current bar for USDCHF H1: ",iTime("USDCHF",PERIOD_H1,i),", ",
iOpen("USDCHF",PERIOD_H1,i),", ",
                                      iHigh("USDCHF",PERIOD_H1,i),", ",
iLow("USDCHF",PERIOD_H1,i),", ",
                                      iClose("USDCHF",PERIOD_H1,i),", ",
iVolume("USDCHF",PERIOD_H1,i));
```

```
double iLow(string symbol, int timeframe, int shift)
```

Returns **Low** value for the bar of indicated *symbol* with *timeframe* and *shift*. If local history is empty (not loaded), function returns 0.

Parameters

symbol	-	Symbol the data of which should be used to calculate indicator. NULL means the current symbol.
timeframe	-	Time frame. It can be any of Time frame enumeration values.
shift	-	Shift relative to the current bar (number of periods back), where the data should be taken from.

Sample

```
  Print("Current bar for USDCHF H1: ",iTime("USDCHF",PERIOD_H1,i),", ",
iOpen("USDCHF",PERIOD_H1,i),", ",
                                      iHigh("USDCHF",PERIOD_H1,i),", ",
iLow("USDCHF",PERIOD_H1,i),", ",
                                      iClose("USDCHF",PERIOD_H1,i),", ",
iVolume("USDCHF",PERIOD_H1,i));
```

```
double iOpen(string symbol, int timeframe, int shift)
```

Returns **Open** value for the bar of indicated *symbol* with *timeframe* and *shift*. If local history is empty (not loaded), function returns 0.

Parameters

symbol	-	Symbol the data of which should be used to calculate indicator. NULL means the current symbol.
timeframe	-	Time frame. It can be any of Time frame enumeration values.
shift	-	Shift relative to the current bar (number of periods back), where the data should be taken from.

Sample

```
  Print("Current bar for USDCHF H1: ",iTime("USDCHF",PERIOD_H1,i),", ",
iOpen("USDCHF",PERIOD_H1,i),", ",
                                      iHigh("USDCHF",PERIOD_H1,i),", ",
iLow("USDCHF",PERIOD_H1,i),", ",
                                      iClose("USDCHF",PERIOD_H1,i),", ",
iVolume("USDCHF",PERIOD_H1,i));
```

```
datetime iTime(string symbol, int timeframe, int shift)
```

Returns **Time** value for the bar of indicated *symbol* with *timeframe* and *shift*. If local history is empty (not loaded), function returns 0.

Parameters

symbol	-	Symbol the data of which should be used to calculate indicator. NULL means the current symbol.
timeframe	-	Time frame. It can be any of Time frame enumeration values.
shift	-	Shift relative to the current bar (number of periods back), where the data should be taken from.

Sample

```
  Print("Current bar for USDCHF H1: ",iTime("USDCHF",PERIOD_H1,i),", ",
iOpen("USDCHF",PERIOD_H1,i),", ",
                                      iHigh("USDCHF",PERIOD_H1,i),", ",
iLow("USDCHF",PERIOD_H1,i),", ",
                                      iClose("USDCHF",PERIOD_H1,i),", ",
iVolume("USDCHF",PERIOD_H1,i));
```

```
double iVolume(string symbol, int timeframe, int shift)
```

Returns **Volume** value for the bar of indicated *symbol* with *timeframe* and *shift*. If local history is empty (not loaded), function returns 0.

Parameters

symbol	-	Symbol the data of which should be used to calculate indicator. NULL means the current symbol.
timeframe	-	Time frame. It can be any of Time frame enumeration values.
shift	-	Shift relative to the current bar (number of periods back), where the data should be taken from.

Sample

```
  Print("Current bar for USDCHF H1: ",iTime("USDCHF",PERIOD_H1,i),", ",
iOpen("USDCHF",PERIOD_H1,i),", ",
                                      iHigh("USDCHF",PERIOD_H1,i),", ",
iLow("USDCHF",PERIOD_H1,i),", ",
                                      iClose("USDCHF",PERIOD_H1,i),", ",
iVolume("USDCHF",PERIOD_H1,i));
```

int Highest(string symbol, int timeframe, int type, int count=WHOLE_ARRAY, int start=0)

Returns the shift of the maximum value over a specific number of periods depending on type.

Parameters

symbol	-	Symbol the data of which should be used to calculate indicator. NULL means the current symbol.
timeframe	-	Time frame. It can be any of Time frame enumeration values.
type	-	Series array identifier. It can be any of the Series array identifier enumeration values.
count	-	Number of periods (in direction from the start bar to the back one) on which the calculation is carried out.
start	-	Shift showing the bar, relative to the current bar, that the data should be taken from.

Sample

```
  double val;
  // calculating the highest value in the range from 5 element to 25 element
  // indicator charts symbol and indicator charts time frame
  val=High[Highest(NULL,0,MODE_HIGH,20,4)];
```

int Lowest(string symbol, int timeframe, int type, int count=WHOLE_ARRAY, int start=0)

Returns the shift of the least value over a specific number of periods depending on type.

Parameters

symbol	-	Symbol the data of which should be used to calculate indicator. NULL means the current symbol.
timeframe	-	Time frame. It can be any of Time frame enumeration values.
type	-	Series array identifier. It can be any of Series array identifier enumeration values.
count	-	Number of periods (in direction from the start bar to the back one) on which the calculation is carried out.
start	-	Shift showing the bar, relative to the current bar, that the data should be taken from.

Sample

```
  double val=Low[Lowest(NULL,0,MODE_LOW,10,10)];
```

Trading functions

OrdersTotal()
OrderSwap()
OrderSymbol()
OrderTakeProfit()
OrderTicket()
OrderType()

int HistoryTotal()

Returns the number of closed orders in the account history loaded into the terminal. To get the detailed error information, call GetLastError() function.

Sample

```
// retrieving info from trade history
int i,hstTotal=HistoryTotal();
for(i=0;i<hstTotal;i++)
  {
   //---- check selection result
   if(OrderSelect(i,SELECT_BY_POS,MODE_HISTORY)==false)
     {
      Print("Access to history failed with error (",GetLastError(),")");
      break;
     }
   // some work with order
  }
```

bool OrderClose(int ticket, double lots, double price, int slippage, color Color=CLR_NONE)

Closes opened order. If the function succeeds, the return value is true. If the function fails, the return value is false. To get the detailed error information, call GetLastError().

Parameters

ticket	-	Unique number of the order ticket.
lots	-	Number of lots.
price	-	Preferred closing price.
slippage	-	Value of the maximum price slippage in points.
Color	-	Color of the closing arrow on the chart.If the parameter is absent or has CLR_NONE value closing arrow will not be drawn on the chart.

Sample

```
if(iRSI(NULL,0,14,PRICE_CLOSE,0)>75)
  {
   OrderClose(order_id,1,Ask,3,Red);
   return(0);
  }
```

bool OrderCloseBy(int ticket, int opposite, color Color=CLR_NONE)

Closes opened order by another opposite opened order. If the function succeeds, the return value is true. If the function fails, the return value is false. To get the detailed error information, call GetLastError().

Parameters

ticket	-	Unique number of the order ticket.
opposite	-	Unique number of the opposite order ticket.
Color	-	Color of the closing arrow on the chart.If the parameter is absent or has CLR_NONE value closing arrow will not be drawn on the chart.

Sample

```
if(iRSI(NULL,0,14,PRICE_CLOSE,0)>75)
  {
   OrderCloseBy(order_id,opposite_id);
   return(0);
  }
```

double OrderClosePrice()

Returns close price for the currently selected order.
Note: Order must be selected by OrderSelect() function.

Sample

```
 if(OrderSelect(ticket,SELECT_BY_POS)==true)
   Print("Close price for the order ",ticket," = ",OrderClosePrice());
 else
   Print("OrderSelect failed error code is",GetLastError());
```

datetime OrderCloseTime()

Returns close time for the currently selected order. If order close time is not 0 then the order selected and has been closed and retrieved from the account history.
Note: Order must be selected by OrderSelect() function.

Sample

```
 if(OrderSelect(10,SELECT_BY_POS,MODE_HISTORY)==true)
   {
    datetime ctm=OrderOpenTime();
    if(ctm>0) Print("Open time for the order 10 ", ctm);
    ctm=OrderCloseTime();
    if(ctm>0) Print("Close time for the order 10 ", ctm);
   }
 else
   Print("OrderSelect failed error code is",GetLastError());
```

string OrderComment()

Returns comment for the selected order.
Note: Order must be selected by OrderSelect() function.

Sample

```
 string comment;
 if(OrderSelect(10,SELECT_BY_TICKET)==false)
   {
    Print("OrderSelect failed error code is",GetLastError());
    return(0);
   }
 comment = OrderComment();
 // ...
```

double OrderCommission()

Returns calculated commission for the currently selected order.
Note: Order must be selected by OrderSelect() function.

Sample

```
 if(OrderSelect(10,SELECT_BY_POS)==true)
   Print("Commission for the order 10 ",OrderCommission());
 else
   Print("OrderSelect failed error code is",GetLastError());
```

bool OrderDelete(int ticket)

Deletes previously opened pending order. If the function succeeds, the return value is true. If the function fails, the return value is false. To get the detailed error information, call GetLastError().

Parameters

ticket	-	Unique number of the order ticket.

Sample

```
if(Ask>var1)
  {
   OrderDelete(order_ticket);
   return(0);
  }
```

datetime OrderExpiration()

Returns expiration date for the selected pending order.
Note: Order must be selected by OrderSelect() function.

Sample

```
if(OrderSelect(10, SELECT_BY_TICKET)==true)
  Print("Order expiration for the order #10 is ",OrderExpiration());
else
  Print("OrderSelect failed error code is",GetLastError());
```

double OrderLots()

Returns lots value for the selected order.
Note: Order must be selected by OrderSelect() function.

Sample

```
if(OrderSelect(10,SELECT_BY_POS)==true)
  Print("lots for the order 10 ",OrderLots());
else
  Print("OrderSelect failed error code is",GetLastError());
```

int OrderMagicNumber()

Returns magic number for the currently selected order.
Note: Order must be selected by OrderSelect() function.

Sample

```
if(OrderSelect(10,SELECT_BY_POS)==true)
  Print("Magic number for the order 10 ", OrderMagicNumber());
else
  Print("OrderSelect failed error code is",GetLastError());
```

bool OrderModify(int ticket, double price, double stoploss, double takeprofit, datetime expiration, color arrow_color=CLR_NONE)

Modification of characteristics for the previously opened position or a pending order. If the function succeeds, the return value is true. If the function fails, the return value is false. To get the detailed error information, call GetLastError().

Parameters

ticket	-	Unique number of the order ticket.
price	-	New price (for pending orders only).
stoploss	-	New stoploss level.
takeprofit	-	New profit-taking level.
expiration	-	Order expiration server date/time (for pending orders only).
arrow_color	-	Arrow color of the pictogram on the chart.If the parameter is absent or has CLR_NONE value arrow will not be drawn on the chart.

Sample

```
  if(TrailingStop>0)
    {
     SelectOrder(12345,SELECT_BY_TICKET);
     if(Bid-OrderOpenPrice()>Point*TrailingStop)
       {
        if(OrderStopLoss()<Bid-Point*TrailingStop)
          {
           OrderModify(OrderTicket(),Ask-10*Point,Ask-
35*Point,OrderTakeProfit(),0,Blue);
           return(0);
          }
       }
    }
```

double OrderOpenPrice()

Returns open price for the currently selected order.
Note: Order must be selected by OrderSelect() function.

Sample

```
  if(OrderSelect(10, SELECT_BY_POS)==true)
    Print("open price for the order 10 ",OrderOpenPrice());
  else
    Print("OrderSelect failed error code is",GetLastError());
```

datetime OrderOpenTime()

Returns open time for the currently selected order.
Note: Order must be selected by OrderSelect() function.

Sample

```
  if(OrderSelect(10, SELECT_BY_POS)==true)
    Print("open time for the order 10 ",OrderOpenTime());
  else
    Print("OrderSelect failed error code is",GetLastError());
```

void OrderPrint()

Prints selected order data to the log for the selected order.
Note: Order must be selected by OrderSelect() function.

Sample

```
  if(OrderSelect(10, SELECT_BY_TICKET)==true)
    OrderPrint();
  else
    Print("OrderSelect failed error code is",GetLastError());
```

double OrderProfit()

Returns profit for the currently selected order.

Note: Order must be selected by OrderSelect() function.

Sample

```
 if(OrderSelect(10, SELECT_BY_POS)==true)
   Print("Profit for the order 10 ",OrderProfit());
 else
   Print("OrderSelect failed error code is",GetLastError());
```

bool OrderSelect(int index, int select, int pool=MODE_TRADES)

Selects order by index or ticket to further processing. If the function fails, the return value will be false. To get the extended error information, call GetLastError().

Parameters

index	-	Order index or order ticket depending from second parameter.
select	-	Selecting flags. It can be any of the following values: SELECT_BY_POS - index in the order pool, SELECT_BY_TICKET - index is order ticket.
pool	-	Optional order pool index. Used when select parameter is SELECT_BY_POS.It can be any of the following values: MODE_TRADES (default)- order selected from trading pool(opened and pending orders), MODE_HISTORY - order selected from history pool (closed and canceled order).

Sample

```
 if(OrderSelect(12470, SELECT_BY_TICKET)==true)
   {
    Print("order #12470 open price is ", OrderOpenPrice());
    Print("order #12470 close price is ", OrderClosePrice());
   }
 else
   Print("OrderSelect failed error code is",GetLastError());
```

int OrderSend(string symbol, int cmd, double volume, double price, int slippage, double stoploss, double takeprofit, string comment=NULL, int magic=0, datetime expiration=0, color arrow_color=CLR_NONE)

Main function used to open a position or set a pending order. Returns ticket of the placed order or -1 if failed. To check error code use GetLastError() function.

Parameters

symbol	-	Symbol for trading.
cmd	-	Operation type. It can be any of the Trade operation enumeration.
volume	-	Number of lots.
price	-	Preferred price of the trade.
slippage	-	Maximum price slippage for buy or sell orders.
stoploss	-	Stop loss level.
takeprofit	-	Take profit level.
comment	-	Order comment text. Last part of the comment may be changed by server.
magic	-	Order magic number. May be used as user defined identifier.
expiration	-	Order expiration time (for pending orders only).
arrow_color	-	Color of the opening arrow on the chart. If parameter is absent or has CLR_NONE value opening arrow is not drawn on the chart.

Sample

```
 int ticket;
 if(iRSI(NULL,0,14,PRICE_CLOSE,0)<25)
   {
    ticket=OrderSend(Symbol(),OP_BUY,1,Ask,3,Ask-25*Point,Ask+25*Point,"My order
#2",16384,0,Green);
    if(ticket<0)
      {
       Print("OrderSend failed with error #",GetLastError());
       return(0);
      }
   }
```

double OrderStopLoss()

Returns stop loss value for the currently selected order.
Note: Order must be selected by OrderSelect() function.

Sample

```
  if(OrderSelect(ticket,SELECT_BY_POS)==true)
    Print("Stop loss value for the order 10 ", OrderStopLoss());
  else
    Print("OrderSelect failed error code is",GetLastError());
```

int OrdersTotal()

Returns market and pending orders count.

Sample

```
  int handle=FileOpen("OrdersReport.csv",FILE_WRITE|FILE_CSV,"\t");
  if(handle<0) return(0);
  // write header
  FileWrite(handle,"#","open price","open time","symbol","lots");
  int total=OrdersTotal();
  // write open orders
  for(int pos=0;pos<total;pos++)
    {
     if(OrderSelect(pos,SELECT_BY_POS)==false) continue;
     FileWrite(handle,OrderTicket(),OrderOpenPrice(),OrderOpenTime(),OrderSymbol(),OrderLots());
    }
  FileClose(handle);
```

double OrderSwap()

Returns swap value for the currently selected order.
Note: Order must be selected by OrderSelect() function.

Sample

```
  if(OrderSelect(order_id, SELECT_BY_TICKET)==true)
    Print("Swap for the order #", order_id, " ",OrderSwap());
  else
    Print("OrderSelect failed error code is",GetLastError());
```

string OrderSymbol()

Returns the order symbol value for selected order.
Note: Order must be selected by OrderSelect() function.

Sample

```
  if(OrderSelect(12, SELECT_BY_POS)==true)
    Print("symbol of order #", OrderTicket(), " is ", OrderSymbol());
  else
    Print("OrderSelect failed error code is",GetLastError());
```

double OrderTakeProfit()

Returns take profit value for the currently selected order.

Note: Order must be selected by OrderSelect() function.

Sample

```
if(OrderSelect(12, SELECT_BY_POS)==true)
  Print("Order #",OrderTicket()," profit: ", OrderTakeProfit());
else
  Print("OrderSelect() ▯▯▯ ▯▯▯ - ",GetLastError());
```

`int OrderTicket()`

Returns ticket number for the currently selected order.
Note: Order must be selected by OrderSelect() function.

Sample

```
if(OrderSelect(12, SELECT_BY_POS)==true)
  order=OrderTicket();
else
  Print("OrderSelect failed error code is",GetLastError());
```

`int OrderType()`

Returns order operation type for the currently selected order. It can be any of the following values:
OP_BUY - buying position,
OP_SELL - selling position,
OP_BUYLIMIT - buy limit pending position,
OP_BUYSTOP - buy stop pending position,
OP_SELLLIMIT - sell limit pending position,
OP_SELLSTOP - sell stop pending position.
Note: Order must be selected by OrderSelect() function.

Sample

```
int order_type;
if(OrderSelect(12, SELECT_BY_POS)==true)
  {
   order_type=OrderType();
   // ...
  }
else
  Print("OrderSelect() returned error - ",GetLastError());
```

Window functions

- BarsPerWindow()
- FirstVisibleBar()
- PriceOnDropped()
- TimeOnDropped()
- WindowFind()
- WindowHandle()
- WindowIsVisible
- WindowOnDropped()
- WindowsTotal()
- WindowXOnDropped()
- WindowYOnDropped()

`int BarsPerWindow()`

Function returns the amount of bars visible on the chart.

Sample

```
   // work with visible bars.
   int bars_count=BarsPerWindow();
   int bar=FirstVisibleBar();
   for(int i=0; i<bars_count; i++,bar--)
     {
      // ...
     }
```

int FirstVisibleBar()

Function returns index of the first visible bar.

Sample

```
   // work with visible bars.
   int bars_count=BarsPerWindow();
   int bar=FirstVisibleBar();
   for(int i=0; i<bars_count; i++,bar--)
     {
      // ...
     }
```

int FirstVisibleBar()

Function returns index of the first visible bar.

Sample

```
   // work with visible bars.
   int bars_count=BarsPerWindow();
   int bar=FirstVisibleBar();
   for(int i=0; i<bars_count; i++,bar--)
     {
      // ...
     }
```

datetime TimeOnDropped()

Returns time part of dropped point where expert or script was dropped. This value is valid when expert or script dropped by mouse.
Note: For custom indicators this value is undefined.

Sample

```
  double   drop_price=PriceOnDropped();
  datetime drop_time=TimeOnDropped();
  //---- may be undefined (zero)
  if(drop_time>0)
    {
     ObjectCreate("Dropped price line", OBJ_HLINE, 0, drop_price);
     ObjectCreate("Dropped time line", OBJ_VLINE, 0, drop_time);
    }
```

int WindowFind(string name)

If indicator with *name* found returns the window index containing specified indicator, otherwise returns -1.
Note: WindowFind() returns -1 if ñustom indicator searches itself when *init()* function works.

Parameters

name	-	Indicator short name.

Sample

```
int win_idx=WindowFind("MACD(12,26,9)");
```

```
int WindowHandle(string symbol, int timeframe)
```

If chart of *symbol* and *timeframe* is currently opened returns the window handle, otherwise returns 0.

Parameters

symbol	-	symbol name.
timeframe	-	Time frame. It can be any of Time frame enumeration values.

Sample

```
int win_handle=WindowHandle("USDX",PERIOD_H1);
if(win_handle!=0)
  Print("Window with USDX,H1 detected. Rates array will be copied immediately.");
```

```
bool WindowIsVisible(int index)
```

Returns true if the chart subwindow is visible, otherwise returns false.

Parameters

index	-	Chart subwindow index.

Sample

```
int maywin=WindowFind("MyMACD");
if(maywin>-1 && WindowIsVisible(maywin)==true)
  Print("window of MyMACD is visible");
else
  Print("window of MyMACD not found or is not visible");
```

```
int WindowOnDropped()
```

Returns window index where expert, custom indicator or script was dropped. This value is valid when expert, custom indicator or script dropped by mouse.
Note: For custom indicators this index is undefined when *init()* function works and returning index is window index where custom indicator works (may be different from dropping window index, because custom indicator can create its own new window). **See also** WindowXOnDropped(), WindowYOnDropped()

Sample

```
if(WindowOnDropped()!=0)
  {
   Print("Indicator 'MyIndicator' must be applied to main chart window!");
   return(false);
  }
```

```
int WindowsTotal()
```

Returns count of indicator windows on the chart (including main chart).

Sample

```
Print("Windows count = ", WindowsTotal());
```

```
int WindowXOnDropped()
```

Returns x-axis coordinate in pixels were expert or script dropped to the chart. **See also** WindowYOnDropped(), WindowOnDropped()

Sample

```
Print("Expert dropped point x=",WindowXOnDropped()," y=",WindowYOnDropped());
```

int WindowYOnDropped()

Returns y-axis coordinate in pixels were expert or script dropped to the chart. **See also** WindowYOnDropped(), WindowOnDropped()

Sample

```
Print("Expert dropped point x=",WindowXOnDropped()," y=",WindowYOnDropped());
```

Include Files

```
//+------------------------------------------------------------------+
//|                                                   stderror.mqh |
//|                 Copyright © 2004-2005, MetaQuotes Software Corp. |
//|                                        http://www.metaquotes.net/ |
//+------------------------------------------------------------------+
//---- errors returned from trade server
#define ERR_NO_ERROR                                       0
#define ERR_NO_RESULT                                      1
#define ERR_COMMON_ERROR                                   2
#define ERR_INVALID_TRADE_PARAMETERS                       3
#define ERR_SERVER_BUSY                                    4
#define ERR_OLD_VERSION                                    5
#define ERR_NO_CONNECTION                                  6
#define ERR_NOT_ENOUGH_RIGHTS                              7
#define ERR_TOO_FREQUENT_REQUESTS                          8
#define ERR_MALFUNCTIONAL_TRADE                            9
#define ERR_ACCOUNT_DISABLED                              64
#define ERR_INVALID_ACCOUNT                               65
#define ERR_TRADE_TIMEOUT                                128
#define ERR_INVALID_PRICE                                129
#define ERR_INVALID_STOPS                                130
#define ERR_INVALID_TRADE_VOLUME                         131
#define ERR_MARKET_CLOSED                                132
#define ERR_TRADE_DISABLED                               133
#define ERR_NOT_ENOUGH_MONEY                             134
#define ERR_PRICE_CHANGED                                135
#define ERR_OFF_QUOTES                                   136
#define ERR_BROKER_BUSY                                  137
#define ERR_REQUOTE                                      138
#define ERR_ORDER_LOCKED                                 139
#define ERR_LONG_POSITIONS_ONLY_ALLOWED                  140
#define ERR_TOO_MANY_REQUESTS                            141
#define ERR_TRADE_MODIFY_DENIED                          145
#define ERR_TRADE_CONTEXT_BUSY                           146
//---- mql4 run time errors
#define ERR_NO_MQLERROR                                 4000
#define ERR_WRONG_FUNCTION_POINTER                      4001
#define ERR_ARRAY_INDEX_OUT_OF_RANGE                    4002
#define ERR_NO_MEMORY_FOR_FUNCTION_CALL_STACK           4003
#define ERR_RECURSIVE_STACK_OVERFLOW                    4004
#define ERR_NOT_ENOUGH_STACK_FOR_PARAMETER              4005
#define ERR_NO_MEMORY_FOR_PARAMETER_STRING              4006
#define ERR_NO_MEMORY_FOR_TEMP_STRING                   4007
#define ERR_NOT_INITIALIZED_STRING                      4008
#define ERR_NOT_INITIALIZED_ARRAYSTRING                 4009
#define ERR_NO_MEMORY_FOR_ARRAYSTRING                   4010
#define ERR_TOO_LONG_STRING                             4011
#define ERR_REMAINDER_FROM_ZERO_DIVIDE                  4012
#define ERR_ZERO_DIVIDE                                 4013
#define ERR_UNKNOWN_COMMAND                             4014
#define ERR_WRONG_JUMP                                  4015
#define ERR_NOT_INITIALIZED_ARRAY                       4016
#define ERR_DLL_CALLS_NOT_ALLOWED                       4017
#define ERR_CANNOT_LOAD_LIBRARY                         4018
#define ERR_CANNOT_CALL_FUNCTION                        4019
```

```
#define ERR_EXTERNAL_EXPERT_CALLS_NOT_ALLOWED      4020
#define ERR_NOT_ENOUGH_MEMORY_FOR_RETURNED_STRING  4021
#define ERR_SYSTEM_BUSY                            4022
#define ERR_INVALID_FUNCTION_PARAMETERS_COUNT      4050
#define ERR_INVALID_FUNCTION_PARAMETER_VALUE       4051
#define ERR_STRING_FUNCTION_INTERNAL_ERROR         4052
#define ERR_SOME_ARRAY_ERROR                       4053
#define ERR_INCORRECT_SERIES_ARRAY_USING           4054
#define ERR_CUSTOM_INDICATOR_ERROR                 4055
#define ERR_INCOMPATIBLE_ARRAYS                    4056
#define ERR_GLOBAL_VARIABLES_PROCESSING_ERROR      4057
#define ERR_GLOBAL_VARIABLE_NOT_FOUND              4058
#define ERR_FUNCTION_NOT_ALLOWED_IN_TESTING_MODE   4059
#define ERR_FUNCTION_NOT_CONFIRMED                 4060
#define ERR_SEND_MAIL_ERROR                        4061
#define ERR_STRING_PARAMETER_EXPECTED              4062
#define ERR_INTEGER_PARAMETER_EXPECTED             4063
#define ERR_DOUBLE_PARAMETER_EXPECTED              4064
#define ERR_ARRAY_AS_PARAMETER_EXPECTED            4065
#define ERR_HISTORY_WILL_UPDATED                   4066
#define ERR_END_OF_FILE                            4099
#define ERR_SOME_FILE_ERROR                        4100
#define ERR_WRONG_FILE_NAME                        4101
#define ERR_TOO_MANY_OPENED_FILES                  4102
#define ERR_CANNOT_OPEN_FILE                       4103
#define ERR_INCOMPATIBLE_ACCESS_TO_FILE            4104
#define ERR_NO_ORDER_SELECTED                      4105
#define ERR_UNKNOWN_SYMBOL                         4106
#define ERR_INVALID_PRICE_PARAM                    4107
#define ERR_INVALID_TICKET                         4108
#define ERR_TRADE_NOT_ALLOWED                      4109
#define ERR_LONGS__NOT_ALLOWED                     4110
#define ERR_SHORTS_NOT_ALLOWED                     4111
#define ERR_OBJECT_ALREADY_EXISTS                  4200
#define ERR_UNKNOWN_OBJECT_PROPERTY                4201
#define ERR_OBJECT_DOES_NOT_EXIST                  4202
#define ERR_UNKNOWN_OBJECT_TYPE                    4203
#define ERR_NO_OBJECT_NAME                         4204
#define ERR_OBJECT_COORDINATES_ERROR               4205
#define ERR_NO_SPECIFIED_SUBWINDOW                 4206
```

```
//+------------------------------------------------------------------+
//|                                                       stdlib.mqh |
//|                      Copyright © 2004, MetaQuotes Software Corp. |
//|                                       http://www.metaquotes.net/ |
//+------------------------------------------------------------------+
#import "stdlib.ex4"

string ErrorDescription(int error_code);
int    RGB(int red_value,int green_value,int blue_value);
bool   CompareDoubles(double number1,double number2);
string DoubleToStrMorePrecision(double number,int precision);
string IntegerToHexString(int integer_number);
```

```
//+------------------------------------------------------------------+
//|                                                    WinUser32.mqh |
//|                      Copyright © 2004, MetaQuotes Software Corp. |
//|                                       http://www.metaquotes.net/ |
//+------------------------------------------------------------------+
#define   copyright "Copyright © 2004, MetaQuotes Software Corp."
#define   link      "http://www.metaquotes.net/"

#import "user32.dll"
   //---- messages
   int      SendMessageA(int hWnd,int Msg,int wParam,int lParam);
   int      SendNotifyMessageA(int hWnd,int Msg,int wParam,int lParam);
   int      PostMessageA(int hWnd,int Msg,int wParam,int lParam);
   void     keybd_event(int bVk,int bScan,int dwFlags,int dwExtraInfo);
   void     mouse_event(int dwFlags,int dx,int dy,int dwData,int dwExtraInfo);
   //---- windows
   int      FindWindowA(string lpClassName ,string lpWindowName);
   int      SetWindowTextA(int hWnd,string lpString);
   int      GetWindowTextA(int hWnd,string lpString,int nMaxCount);
   int      GetWindowTextLengthA(int hWnd);
   int      GetWindow(int hWnd,int uCmd);

   int      UpdateWindow(int hWnd);
   int      EnableWindow(int hWnd,int bEnable);
   int      DestroyWindow(int hWnd);
   int      ShowWindow(int hWnd,int nCmdShow);
   int      SetActiveWindow(int hWnd);
   int      AnimateWindow(int hWnd,int dwTime,int dwFlags);
   int      FlashWindow(int hWnd,int dwFlags /*bInvert*/);
   int      CloseWindow(int hWnd);
   int      MoveWindow(int hWnd,int X,int Y,int nWidth,int nHeight,int bRepaint);
   int      SetWindowPos(int hWnd,int hWndInsertAfter ,int X,int Y,int cx,int cy,int
uFlags);
   int      IsWindowVisible(int hWnd);
   int      IsIconic(int hWnd);
   int      IsZoomed(int hWnd);
   int      SetFocus(int hWnd);
   int      GetFocus();
   int      GetActiveWindow();
   int      IsWindowEnabled(int hWnd);
   //---- miscelaneouse
   int      MessageBoxA(int hWnd ,string lpText,string lpCaption,int uType);
   int      MessageBoxExA(int hWnd ,string lpText,string lpCaption,int uType,int
wLanguageId);
   int      MessageBeep(int uType);
   int      GetSystemMetrics(int nIndex);
   int      ExitWindowsEx(int uFlags,int dwReserved);
   int      SwapMouseButton(int fSwap);
#import

//---- Window Messages
#define WM_NULL                       0x0000
#define WM_CREATE                     0x0001
#define WM_DESTROY                    0x0002
#define WM_MOVE                       0x0003
#define WM_SIZE                       0x0005
#define WM_ACTIVATE                   0x0006
#define WM_SETFOCUS                   0x0007
#define WM_KILLFOCUS                  0x0008
#define WM_ENABLE                     0x000A
#define WM_SETREDRAW                  0x000B
#define WM_SETTEXT                    0x000C
#define WM_GETTEXT                    0x000D
#define WM_GETTEXTLENGTH              0x000E
#define WM_PAINT                      0x000F
#define WM_CLOSE                      0x0010
#define WM_QUERYENDSESSION            0x0011
#define WM_QUIT                       0x0012
#define WM_QUERYOPEN                  0x0013
#define WM_ERASEBKGND                 0x0014
#define WM_SYSCOLORCHANGE             0x0015
#define WM_ENDSESSION                 0x0016
#define WM_SHOWWINDOW                 0x0018
#define WM_WININICHANGE               0x001A
#define WM_SETTINGCHANGE              0x001A // WM_WININICHANGE
#define WM_DEVMODECHANGE              0x001B
#define WM_ACTIVATEAPP                0x001C
```

```
#define WM_FONTCHANGE                       0x001D
#define WM_TIMECHANGE                       0x001E
#define WM_CANCELMODE                       0x001F
#define WM_SETCURSOR                        0x0020
#define WM_MOUSEACTIVATE                    0x0021
#define WM_CHILDACTIVATE                    0x0022
#define WM_QUEUESYNC                        0x0023
#define WM_GETMINMAXINFO                    0x0024
#define WM_PAINTICON                        0x0026
#define WM_ICONERASEBKGND                   0x0027
#define WM_NEXTDLGCTL                       0x0028
#define WM_SPOOLERSTATUS                    0x002A
#define WM_DRAWITEM                         0x002B
#define WM_MEASUREITEM                      0x002C
#define WM_DELETEITEM                       0x002D
#define WM_VKEYTOITEM                       0x002E
#define WM_CHARTOITEM                       0x002F
#define WM_SETFONT                          0x0030
#define WM_GETFONT                          0x0031
#define WM_SETHOTKEY                        0x0032
#define WM_GETHOTKEY                        0x0033
#define WM_QUERYDRAGICON                    0x0037
#define WM_COMPAREITEM                      0x0039
#define WM_GETOBJECT                        0x003D
#define WM_COMPACTING                       0x0041
#define WM_WINDOWPOSCHANGING                0x0046
#define WM_WINDOWPOSCHANGED                 0x0047
#define WM_COPYDATA                         0x004A
#define WM_CANCELJOURNAL                    0x004B
#define WM_NOTIFY                           0x004E
#define WM_INPUTLANGCHANGEREQUEST           0x0050
#define WM_INPUTLANGCHANGE                  0x0051
#define WM_TCARD                            0x0052
#define WM_HELP                             0x0053
#define WM_USERCHANGED                      0x0054
#define WM_NOTIFYFORMAT                     0x0055
#define WM_CONTEXTMENU                      0x007B
#define WM_STYLECHANGING                    0x007C
#define WM_STYLECHANGED                     0x007D
#define WM_DISPLAYCHANGE                    0x007E
#define WM_GETICON                          0x007F
#define WM_SETICON                          0x0080
#define WM_NCCREATE                         0x0081
#define WM_NCDESTROY                        0x0082
#define WM_NCCALCSIZE                       0x0083
#define WM_NCHITTEST                        0x0084
#define WM_NCPAINT                          0x0085
#define WM_NCACTIVATE                       0x0086
#define WM_GETDLGCODE                       0x0087
#define WM_SYNCPAINT                        0x0088
#define WM_NCMOUSEMOVE                      0x00A0
#define WM_NCLBUTTONDOWN                    0x00A1
#define WM_NCLBUTTONUP                      0x00A2
#define WM_NCLBUTTONDBLCLK                  0x00A3
#define WM_NCRBUTTONDOWN                    0x00A4
#define WM_NCRBUTTONUP                      0x00A5
#define WM_NCRBUTTONDBLCLK                  0x00A6
#define WM_NCMBUTTONDOWN                    0x00A7
#define WM_NCMBUTTONUP                      0x00A8
#define WM_NCMBUTTONDBLCLK                  0x00A9
#define WM_KEYFIRST                         0x0100
#define WM_KEYDOWN                          0x0100
#define WM_KEYUP                            0x0101
#define WM_CHAR                             0x0102
#define WM_DEADCHAR                         0x0103
#define WM_SYSKEYDOWN                       0x0104
#define WM_SYSKEYUP                         0x0105
```

```
#define WM_SYSCHAR                      0x0106
#define WM_SYSDEADCHAR                  0x0107
#define WM_KEYLAST                      0x0108
#define WM_INITDIALOG                   0x0110
#define WM_COMMAND                      0x0111
#define WM_SYSCOMMAND                   0x0112
#define WM_TIMER                        0x0113
#define WM_HSCROLL                      0x0114
#define WM_VSCROLL                      0x0115
#define WM_INITMENU                     0x0116
#define WM_INITMENUPOPUP                0x0117
#define WM_MENUSELECT                   0x011F
#define WM_MENUCHAR                     0x0120
#define WM_ENTERIDLE                    0x0121
#define WM_MENURBUTTONUP                0x0122
#define WM_MENUDRAG                     0x0123
#define WM_MENUGETOBJECT                0x0124
#define WM_UNINITMENUPOPUP              0x0125
#define WM_MENUCOMMAND                  0x0126
#define WM_CTLCOLORMSGBOX               0x0132
#define WM_CTLCOLOREDIT                 0x0133
#define WM_CTLCOLORLISTBOX              0x0134
#define WM_CTLCOLORBTN                  0x0135
#define WM_CTLCOLORDLG                  0x0136
#define WM_CTLCOLORSCROLLBAR            0x0137
#define WM_CTLCOLORSTATIC               0x0138
#define WM_MOUSEFIRST                   0x0200
#define WM_MOUSEMOVE                    0x0200
#define WM_LBUTTONDOWN                  0x0201
#define WM_LBUTTONUP                    0x0202
#define WM_LBUTTONDBLCLK                0x0203
#define WM_RBUTTONDOWN                  0x0204
#define WM_RBUTTONUP                    0x0205
#define WM_RBUTTONDBLCLK                0x0206
#define WM_MBUTTONDOWN                  0x0207
#define WM_MBUTTONUP                    0x0208
#define WM_MBUTTONDBLCLK                0x0209
#define WM_PARENTNOTIFY                 0x0210
#define WM_ENTERMENULOOP                0x0211
#define WM_EXITMENULOOP                 0x0212
#define WM_NEXTMENU                     0x0213
#define WM_SIZING                       0x0214
#define WM_CAPTURECHANGED               0x0215
#define WM_MOVING                       0x0216
#define WM_DEVICECHANGE                 0x0219
#define WM_MDICREATE                    0x0220
#define WM_MDIDESTROY                   0x0221
#define WM_MDIACTIVATE                  0x0222
#define WM_MDIRESTORE                   0x0223
#define WM_MDINEXT                      0x0224
#define WM_MDIMAXIMIZE                  0x0225
#define WM_MDITILE                      0x0226
#define WM_MDICASCADE                   0x0227
#define WM_MDIICONARRANGE               0x0228
#define WM_MDIGETACTIVE                 0x0229
#define WM_MDISETMENU                   0x0230
#define WM_ENTERSIZEMOVE                0x0231
#define WM_EXITSIZEMOVE                 0x0232
#define WM_DROPFILES                    0x0233
#define WM_MDIREFRESHMENU               0x0234
#define WM_MOUSEHOVER                   0x02A1
#define WM_MOUSELEAVE                   0x02A3
#define WM_CUT                          0x0300
#define WM_COPY                         0x0301
#define WM_PASTE                        0x0302
#define WM_CLEAR                        0x0303
#define WM_UNDO                         0x0304
```

```
#define WM_RENDERFORMAT                    0x0305
#define WM_RENDERALLFORMATS                0x0306
#define WM_DESTROYCLIPBOARD                0x0307
#define WM_DRAWCLIPBOARD                   0x0308
#define WM_PAINTCLIPBOARD                  0x0309
#define WM_VSCROLLCLIPBOARD                0x030A
#define WM_SIZECLIPBOARD                   0x030B
#define WM_ASKCBFORMATNAME                 0x030C
#define WM_CHANGECBCHAIN                   0x030D
#define WM_HSCROLLCLIPBOARD                0x030E
#define WM_QUERYNEWPALETTE                 0x030F
#define WM_PALETTEISCHANGING               0x0310
#define WM_PALETTECHANGED                  0x0311
#define WM_HOTKEY                          0x0312
#define WM_PRINT                           0x0317
#define WM_PRINTCLIENT                     0x0318
#define WM_HANDHELDFIRST                   0x0358
#define WM_HANDHELDLAST                    0x035F
#define WM_AFXFIRST                        0x0360
#define WM_AFXLAST                         0x037F
#define WM_PENWINFIRST                     0x0380
#define WM_PENWINLAST                      0x038F
#define WM_APP                             0x8000

//---- keybd_event routines
#define KEYEVENTF_EXTENDEDKEY              0x0001
#define KEYEVENTF_KEYUP                    0x0002
//---- mouse_event routines
#define MOUSEEVENTF_MOVE                   0x0001 // mouse move
#define MOUSEEVENTF_LEFTDOWN               0x0002 // left button down
#define MOUSEEVENTF_LEFTUP                 0x0004 // left button up
#define MOUSEEVENTF_RIGHTDOWN              0x0008 // right button down
#define MOUSEEVENTF_RIGHTUP                0x0010 // right button up
#define MOUSEEVENTF_MIDDLEDOWN             0x0020 // middle button down
#define MOUSEEVENTF_MIDDLEUP               0x0040 // middle button up
#define MOUSEEVENTF_WHEEL                  0x0800 // wheel button rolled
#define MOUSEEVENTF_ABSOLUTE               0x8000 // absolute move

//---- GetSystemMetrics() codes
#define SM_CXSCREEN                        0
#define SM_CYSCREEN                        1
#define SM_CXVSCROLL                       2
#define SM_CYHSCROLL                       3
#define SM_CYCAPTION                       4
#define SM_CXBORDER                        5
#define SM_CYBORDER                        6
#define SM_CXDLGFRAME                      7
#define SM_CYDLGFRAME                      8
#define SM_CYVTHUMB                        9
#define SM_CXHTHUMB                        10
#define SM_CXICON                          11
#define SM_CYICON                          12
#define SM_CXCURSOR                        13
#define SM_CYCURSOR                        14
#define SM_CYMENU                          15
#define SM_CXFULLSCREEN                    16
#define SM_CYFULLSCREEN                    17
#define SM_CYKANJIWINDOW                   18
#define SM_MOUSEPRESENT                    19
#define SM_CYVSCROLL                       20
#define SM_CXHSCROLL                       21
#define SM_DEBUG                           22
#define SM_SWAPBUTTON                      23
#define SM_RESERVED1                       24
#define SM_RESERVED2                       25
#define SM_RESERVED3                       26
#define SM_RESERVED4                       27
```

```
#define SM_CXMIN                        28
#define SM_CYMIN                        29
#define SM_CXSIZE                       30
#define SM_CYSIZE                       31
#define SM_CXFRAME                      32
#define SM_CYFRAME                      33
#define SM_CXMINTRACK                   34
#define SM_CYMINTRACK                   35
#define SM_CXDOUBLECLK                  36
#define SM_CYDOUBLECLK                  37
#define SM_CXICONSPACING                38
#define SM_CYICONSPACING                39
#define SM_MENUDROPALIGNMENT            40
#define SM_PENWINDOWS                   41
#define SM_DBCSENABLED                  42
#define SM_CMOUSEBUTTONS                43
#define SM_SECURE                       44
#define SM_CXEDGE                       45
#define SM_CYEDGE                       46
#define SM_CXMINSPACING                 47
#define SM_CYMINSPACING                 48
#define SM_CXSMICON                     49
#define SM_CYSMICON                     50
#define SM_CYSMCAPTION                  51
#define SM_CXSMSIZE                     52
#define SM_CYSMSIZE                     53
#define SM_CXMENUSIZE                   54
#define SM_CYMENUSIZE                   55
#define SM_ARRANGE                      56
#define SM_CXMINIMIZED                  57
#define SM_CYMINIMIZED                  58
#define SM_CXMAXTRACK                   59
#define SM_CYMAXTRACK                   60
#define SM_CXMAXIMIZED                  61
#define SM_CYMAXIMIZED                  62
#define SM_NETWORK                      63
#define SM_CLEANBOOT                    67
#define SM_CXDRAG                       68
#define SM_CYDRAG                       69
#define SM_SHOWSOUNDS                   70
#define SM_CXMENUCHECK                  71 // Use instead of
GetMenuCheckMarkDimensions()!
#define SM_CYMENUCHECK                  72
#define SM_SLOWMACHINE                  73
#define SM_MIDEASTENABLED               74
#define SM_MOUSEWHEELPRESENT            75
#define SM_XVIRTUALSCREEN               76
#define SM_YVIRTUALSCREEN               77
#define SM_CXVIRTUALSCREEN              78
#define SM_CYVIRTUALSCREEN              79
#define SM_CMONITORS                    80
#define SM_SAMEDISPLAYFORMAT            81

//---- GetWindow() Constants
#define GW_HWNDFIRST                    0
#define GW_HWNDLAST                     1
#define GW_HWNDNEXT                     2
#define GW_HWNDPREV                     3
#define GW_OWNER                        4
#define GW_CHILD                        5

//---- AnimateWindow() Commands
#define AW_HOR_POSITIVE                 0x00000001
#define AW_HOR_NEGATIVE                 0x00000002
#define AW_VER_POSITIVE                 0x00000004
#define AW_VER_NEGATIVE                 0x00000008
#define AW_CENTER                       0x00000010
```

```
#define AW_HIDE                       0x00010000
#define AW_ACTIVATE                   0x00020000
#define AW_SLIDE                      0x00040000
#define AW_BLEND                      0x00080000

//---- MessageBox() Flags
#define MB_OK                            0x00000000
#define MB_OKCANCEL                      0x00000001
#define MB_ABORTRETRYIGNORE              0x00000002
#define MB_YESNOCANCEL                   0x00000003
#define MB_YESNO                         0x00000004
#define MB_RETRYCANCEL                   0x00000005
#define MB_ICONHAND                      0x00000010
#define MB_ICONQUESTION                  0x00000020
#define MB_ICONEXCLAMATION               0x00000030
#define MB_ICONASTERISK                  0x00000040
#define MB_USERICON                      0x00000080
#define MB_ICONWARNING                   MB_ICONEXCLAMATION
#define MB_ICONERROR                     MB_ICONHAND
#define MB_ICONINFORMATION               MB_ICONASTERISK
#define MB_ICONSTOP                      MB_ICONHAND
#define MB_DEFBUTTON1                    0x00000000
#define MB_DEFBUTTON2                    0x00000100
#define MB_DEFBUTTON3                    0x00000200
#define MB_DEFBUTTON4                    0x00000300
#define MB_APPLMODAL                     0x00000000
#define MB_SYSTEMMODAL                   0x00001000
#define MB_TASKMODAL                     0x00002000
#define MB_HELP                          0x00004000 // Help Button
#define MB_NOFOCUS                       0x00008000
#define MB_SETFOREGROUND                 0x00010000
#define MB_DEFAULT_DESKTOP_ONLY          0x00020000
#define MB_TOPMOST                       0x00040000
#define MB_RIGHT                         0x00080000
#define MB_RTLREADING                    0x00100000

//---- Dialog Box Command IDs
#define IDOK                          1
#define IDCANCEL                      2
#define IDABORT                       3
#define IDRETRY                       4
#define IDIGNORE                      5
#define IDYES                         6
#define IDNO                          7
#define IDCLOSE                       8
#define IDHELP                        9

//+------------------------------------------------------------------+
```

Scripts

```
//+------------------------------------------------------------------+
//|                                             Period_Converter.mq4 |
//|                      Copyright © 2005, MetaQuotes Software Corp. |
//|                                       http://www.metaquotes.net |
//+------------------------------------------------------------------+
#property copyright "Copyright © 2005, MetaQuotes Software Corp."
#property link      "http://www.metaquotes.net"
#property show_inputs
#include <WinUser32.mqh>

extern int ExtPeriodMultiplier=3; // new period multiplier factor
int        ExtHandle=-1;
//+------------------------------------------------------------------+
//| script program start function                                    |
//+------------------------------------------------------------------+
int start()
  {
   int    i, start_pos, i_time, time0, last_fpos, periodseconds;
   double d_open, d_low, d_high, d_close, d_volume, last_volume;
   int    hwnd=0,cnt=0;
//---- History header
   int    version=400;
   string c_copyright;
   string c_symbol=Symbol();
   int    i_period=Period()*ExtPeriodMultiplier;
   int    i_digits=Digits;
   int    i_unused[13];
//----
   ExtHandle=FileOpenHistory(c_symbol+i_period+".hst", FILE_BIN|FILE_WRITE);
   if(ExtHandle < 0) return(-1);
//---- write history file header
   c_copyright="(C)opyright 2003, MetaQuotes Software Corp.";
   FileWriteInteger(ExtHandle, version, LONG_VALUE);
   FileWriteString(ExtHandle, c_copyright, 64);
   FileWriteString(ExtHandle, c_symbol, 12);
   FileWriteInteger(ExtHandle, i_period, LONG_VALUE);
   FileWriteInteger(ExtHandle, i_digits, LONG_VALUE);
   FileWriteInteger(ExtHandle, 0, LONG_VALUE);       //timesign
   FileWriteInteger(ExtHandle, 0, LONG_VALUE);       //last_sync
   FileWriteArray(ExtHandle, i_unused, 0, 13);
//---- write history file
   periodseconds=i_period*60;
   start_pos=Bars-1;
   d_open=Open[start_pos];
   d_low=Low[start_pos];
   d_high=High[start_pos];
   d_volume=Volume[start_pos];
   //---- normalize open time
   i_time=Time[start_pos]/periodseconds;
   i_time*=periodseconds;
   for(i=start_pos-1;i>=0; i--)
     {
      time0=Time[i];
      if(time0>=i_time+periodseconds || i==0)
        {
         if(i==0 && time0<i_time+periodseconds)
           {
            d_volume+=Volume[0];
            if (Low[0]<d_low)   d_low=Low[0];
            if (High[0]>d_high) d_high=High[0];
            d_close=Close[0];
           }
         last_fpos=FileTell(ExtHandle);
         last_volume=Volume[i];
         FileWriteInteger(ExtHandle, i_time, LONG_VALUE);
         FileWriteDouble(ExtHandle, d_open, DOUBLE_VALUE);
         FileWriteDouble(ExtHandle, d_low, DOUBLE_VALUE);
         FileWriteDouble(ExtHandle, d_high, DOUBLE_VALUE);
         FileWriteDouble(ExtHandle, d_close, DOUBLE_VALUE);
         FileWriteDouble(ExtHandle, d_volume, DOUBLE_VALUE);
         FileFlush(ExtHandle);
         cnt++;
         if(time0>=i_time+periodseconds)
           {
            i_time=time0/periodseconds;
            i_time*=periodseconds;
            d_open=Open[i];
            d_low=Low[i];
            d_high=High[i];
            d_close=Close[i];
            d_volume=last_volume;
           }
        }
       else
        {
         d_volume+=Volume[i];
         if (Low[i]<d_low)   d_low=Low[i];
         if (High[i]>d_high) d_high=High[i];
         d_close=Close[i];
        }
     }
   FileFlush(ExtHandle);
   Print(cnt," record(s) written");
//---- collect incoming ticks
   int last_time=LocalTime()-5;
   while(true)
     {
      int cur_time=LocalTime();
      //---- check for new rates
      if(RefreshRates())
        {
         time0=Time[0];
         FileSeek(ExtHandle,last_fpos,SEEK_SET);
         //---- is there current bar?
         if(time0<i_time+periodseconds)
           {
            d_volume+=Volume[0]-last_volume;
            last_volume=Volume[0];
            if (Low[0]<d_low) d_low=Low[0];
            if (High[0]>d_high) d_high=High[0];
            d_close=Close[0];
           }
         else
           {
            //---- no, there is new bar
            d_volume+=Volume[1]-last_volume;
            if (Low[1]<d_low) d_low=Low[1];
            if (High[1]>d_high) d_high=High[1];
            //---- write previous bar remains
            FileWriteInteger(ExtHandle, i_time, LONG_VALUE);
            FileWriteDouble(ExtHandle, d_open, DOUBLE_VALUE);
            FileWriteDouble(ExtHandle, d_low, DOUBLE_VALUE);
            FileWriteDouble(ExtHandle, d_high, DOUBLE_VALUE);
            FileWriteDouble(ExtHandle, d_close, DOUBLE_VALUE);
            FileWriteDouble(ExtHandle, d_volume, DOUBLE_VALUE);
            last_fpos=FileTell(ExtHandle);
            //----
            i_time=time0/periodseconds;
            i_time*=periodseconds;
            d_open=Open[0];
            d_low=Low[0];
            d_high=High[0];
            d_close=Close[0];
            d_volume=Volume[0];
            last_volume=d_volume;
           }
         //----
         FileWriteInteger(ExtHandle, i_time, LONG_VALUE);
         FileWriteDouble(ExtHandle, d_open, DOUBLE_VALUE);
         FileWriteDouble(ExtHandle, d_low, DOUBLE_VALUE);
         FileWriteDouble(ExtHandle, d_high, DOUBLE_VALUE);
         FileWriteDouble(ExtHandle, d_close, DOUBLE_VALUE);
         FileWriteDouble(ExtHandle, d_volume, DOUBLE_VALUE);
         FileFlush(ExtHandle);
         //----
         if(hwnd==0)
           {
            hwnd=WindowHandle(Symbol(),i_period);
            if(hwnd!=0) Print("Chart window detected");
           }
         //---- refresh window not frequently than 1 time in 2 seconds
         if(hwnd!=0 && cur_time-last_time>=2)
           {
            PostMessageA(hwnd,WM_COMMAND,33324,0);
            last_time=cur_time;
           }
        }
     }
//----
   return(0);
  }
//+------------------------------------------------------------------+
//|                                                                  |
//+------------------------------------------------------------------+
void deinit()
  {
   if(ExtHandle>=0) { FileClose(ExtHandle); ExtHandle=-1; }
  }
//+------------------------------------------------------------------+
```

```
//+------------------------------------------------------------------+
//|                                                    rotate_text.mq4 |
//|                      Copyright © 2004, MetaQuotes Software Corp. |
//|                                       http://www.metaquotes.net/ |
//+------------------------------------------------------------------+
#property copyright "Copyright © 2004, MetaQuotes Software Corp."
#property link      "http://www.metaquotes.net/"
#include <stdlib.mqh>

string line_name="rotating_line";
string object_name1="rotating_text";

void init()
  {
   Print("point = ", Point,"   bars=",Bars);
  }
//+------------------------------------------------------------------+
//|                                                                  |
//+------------------------------------------------------------------+
void deinit()
  {
   ObjectDelete(line_name);
   ObjectDelete(object_name1);
   ObjectsRedraw();
  }
//+------------------------------------------------------------------+
//| script program start function                                    |
//+------------------------------------------------------------------+
int start()
  {
   int    time2;
   int    error,index,fontsize=10;
   double price,price1,price2;
   double angle=0.0;
//----
   price2=High[10]+Point*10;
   ObjectCreate(line_name, OBJ_TRENDBYANGLE, 0, Time[10], price2);
   index=20;
   ObjectCreate(object_name1, OBJ_TEXT, 0, Time[index], Low[index]-Point*100);
   ObjectSetText(object_name1, "rotating_text", fontsize);
   while(IsStopped()==false)
     {
      index++;
      price=ObjectGet(object_name1, OBJPROP_PRICE1)+Point;
      error=GetLastError();
      if(error!=0)
```

```
        {
         Print(object_name1," : ",ErrorDescription(error));
         break;
        }
      ObjectMove(object_name1, 0, Time[index], price);
      ObjectSet(object_name1, OBJPROP_ANGLE, angle*2);
      ObjectSet(object_name1, OBJPROP_FONTSIZE, fontsize);
      ObjectSet(line_name, OBJPROP_WIDTH, angle/18.0);
      double line_angle=360.0-angle;
      if(line_angle==90.0)  ObjectSet(line_name, OBJPROP_PRICE2, price2+Point*50);
      if(line_angle==270.0) ObjectSet(line_name, OBJPROP_PRICE2, price2-Point*50);
      time2=ObjectGet(line_name,OBJPROP_TIME2);
      if(line_angle>90.0 && line_angle<270.0) time2=Time[index+10];
      else                                    time2=Time[0];
      ObjectSet(line_name, OBJPROP_TIME2, time2);
      ObjectSet(line_name, OBJPROP_ANGLE, line_angle);
      ObjectsRedraw();
      angle+=3.0;
      if(angle>=360.0) angle=360.0-angle;
      fontsize++;
      if(fontsize>48) fontsize=6;
      Sleep(500);
      price1=ObjectGetValueByShift(line_name, index);
      if(GetLastError()==0)
        {
         if(MathAbs(price1-price) < Point*50)
           {
            Print("price=",price,"  price1=", price1);
            ObjectSetText(object_name1, "REMOVED", 48, "Arial", RGB(255,215,0));
            ObjectsRedraw();
            Sleep(5000);
//            ObjectDelete(object_name1);
           }
        }
     }
   return(0);
  }
//+------------------------------------------------------------------+
```

```
//+------------------------------------------------------------------+
//|                                                        trade.mq4 |
//|                      Copyright © 2004, MetaQuotes Software Corp. |
//|                                       http://www.metaquotes.net/ |
//+------------------------------------------------------------------+
#property copyright "Copyright © 2004, MetaQuotes Software Corp."
#property link      "http://www.metaquotes.net/"

#include <stdlib.mqh>
#include <WinUser32.mqh>
//+------------------------------------------------------------------+
//| script "trading for all money"                                   |
//+------------------------------------------------------------------+
int start()
  {
//----
   if(MessageBox("Do you really want to BUY 1.00 "+Symbol()+" at ASK price?    ",
                 "Script",MB_YESNO|MB_ICONQUESTION)!=IDYES) return(1);
//----
   int ticket=OrderSend(Symbol(),OP_BUY,1.0,Ask,3,0,0,"expert comment",255,0,CLR_NONE);
   if(ticket<1)
     {
      int error=GetLastError();
      Print("Error = ",ErrorDescription(error));
      return;
     }
//----
   OrderPrint();
   return(0);
  }
//+------------------------------------------------------------------+
```

Templates

```
<expert>
type=INDICATOR_ADVISOR
description=Accelerator Oscilator
separate_window=1
used_buffers=4
<ind>
color=Green
type=DRAW_HISTOGRAM
</ind>
<ind>
color=Red
type=DRAW_HISTOGRAM
</ind>
</expert>
#header#
#property copyright "#copyright#"
#property link      "#link#"

#indicator_properties#
#extern_variables#
#mapping_buffers#
//---- indicator buffers
double ExtGreenBuffer[];
double ExtRedBuffer[];
double ExtMABuffer[];
double ExtBuffer[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int init()
  {
   #buffers_used#;
//---- indicator buffers mapping
   SetIndexBuffer(0, ExtGreenBuffer);
   SetIndexBuffer(1, ExtRedBuffer);
   SetIndexBuffer(2, ExtMABuffer);
   SetIndexBuffer(3, ExtBuffer);
//---- drawing settings
   #indicators_init#
//----
   IndicatorDigits(6);
   SetIndexDrawBegin(0,38);
   SetIndexDrawBegin(1,38);
//---- name for DataWindow and indicator subwindow label
   IndicatorShortName("AC");
//---- initialization done
   return(0);
  }
//+------------------------------------------------------------------+
//| Accelerator/Decelerator Oscillator                               |
//+------------------------------------------------------------------+
int start()
  {
   int    limit;
   int    counted_bars=IndicatorCounted();
   double prev,current;
//---- check for possible errors
   if(counted_bars<0) return(-1);
//---- last counted bar will be recounted
   if(counted_bars>0) counted_bars--;
   limit=Bars-counted_bars;
//---- macd counted in the 1-st additional buffer
   for(int i=0; i<limit; i++)
      ExtMABuffer[i]=iMA(NULL,0,5,0,MODE_SMA,PRICE_MEDIAN,i)-
iMA(NULL,0,34,0,MODE_SMA,PRICE_MEDIAN,i);
//---- signal line counted in the 2-nd additional buffer
   for(i=0; i<limit; i++)
      ExtBuffer[i]=iMAOnArray(ExtMABuffer,Bars,5,0,MODE_SMA,i);
//---- dispatch values between 2 buffers
   bool up=true;
   for(i=limit-1; i>=0; i--)
     {
      current=ExtMABuffer[i]-ExtBuffer[i];
      prev=ExtMABuffer[i+1]-ExtBuffer[i+1];
      if(current>prev) up=true;
      if(current<prev) up=false;
      if(!up)
        {
         ExtRedBuffer[i]=current;
         ExtGreenBuffer[i]=0.0;
        }
      else
        {
         ExtGreenBuffer[i]=current;
         ExtRedBuffer[i]=0.0;
        }
     }
//---- done
   return(0);
  }
//+------------------------------------------------------------------+
```

```
<expert>
type=INDICATOR_ADVISOR
description=
used_buffers=3
<param>
name=JawsPeriod
type=int
value=13
</param>
<param>
name=JawsShift
type=int
value=8
</param>
<param>
name=TeethPeriod
type=int
value=8
</param>
<param>
name=TeethShift
type=int
value=5
</param>
<param>
name=LipsPeriod
type=int
value=5
</param>
<param>
name=LipsShift
type=int
value=3
</param>
<ind>
```

```
color=Blue
</ind>
<ind>
color=Red
</ind>
<ind>
color=Lime
</ind>
</expert>
#header#
#property copyright "#copyright#"
#property link      "#link#"

#indicator_properties#
#extern_variables#
#mapping_buffers#
//---- indicator buffers
double ExtBlueBuffer[];
double ExtRedBuffer[];
double ExtLimeBuffer[];

//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int init()
  {
   #buffers_used#
//---- line shifts when drawing
   SetIndexShift(0,JawsShift);
   SetIndexShift(1,TeethShift);
   SetIndexShift(2,LipsShift);
//---- first positions skipped when drawing
   SetIndexDrawBegin(0,JawsShift+JawsPeriod);
   SetIndexDrawBegin(1,TeethShift+TeethPeriod);
   SetIndexDrawBegin(2,LipsShift+LipsPeriod);
//---- 3 indicator buffers mapping
   SetIndexBuffer(0,ExtBlueBuffer);
   SetIndexBuffer(1,ExtRedBuffer);
   SetIndexBuffer(2,ExtLimeBuffer);
//---- drawing settings
   #indicators_init#
//---- initialization done
   return(0);
  }
//+------------------------------------------------------------------+
//| Bill Williams' Alligator                                         |
//+------------------------------------------------------------------+
int start()
  {
   int limit;
   int counted_bars=IndicatorCounted();
//---- check for possible errors
   if(counted_bars<0) return(-1);
//---- last counted bar will be recounted
   if(counted_bars>0) counted_bars--;
   limit=Bars-counted_bars;
//---- main loop
   for(int i=0; i<limit; i++)
     {
      //---- ma_shift set to 0 because SetIndexShift called abowe
      ExtBlueBuffer[i]=iMA(NULL,0,JawsPeriod,0,MODE_SMMA,PRICE_MEDIAN,i);
      ExtRedBuffer[i]=iMA(NULL,0,TeethPeriod,0,MODE_SMMA,PRICE_MEDIAN,i);
      ExtLimeBuffer[i]=iMA(NULL,0,LipsPeriod,0,MODE_SMMA,PRICE_MEDIAN,i);
     }
//---- done
   return(0);
  }
```

```
//+------------------------------------------------------------------+
```

```
<expert>
type=INDICATOR_ADVISOR
description=Awesome Oscilator
separate_window=1
used_buffers=3;
<ind>
color=Green
type=DRAW_HISTOGRAM
</ind>
<ind>
color=Red
type=DRAW_HISTOGRAM
</ind>
</expert>
#header#
#property copyright "#copyright#"
#property link      "#link#"

#indicator_properties#
#extern_variables#
#mapping_buffers#
//---- indicator buffers
double ExtGreenBuffer[];
double ExtRedBuffer[];
double ExtMABuffer[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int init()
  {
   #buffers_used#
//---- drawing settings
   #indicators_init#
   IndicatorDigits(5);
   SetIndexDrawBegin(0,34);
   SetIndexDrawBegin(1,34);
//---- indicator buffers mapping
   SetIndexBuffer(0, ExtGreenBuffer);
   SetIndexBuffer(1, ExtRedBuffer);
   SetIndexBuffer(2, ExtMABuffer);
//---- name for DataWindow and indicator subwindow label
   IndicatorShortName("AO");
//---- initialization done
   return(0);
  }
//+------------------------------------------------------------------+
//| Awesome Oscillator                                               |
//+------------------------------------------------------------------+
int start()
  {
   int    limit;
   int    counted_bars=IndicatorCounted();
   double prev,current;
//---- check for possible errors
   if(counted_bars<0) return(-1);
   //---- last counted bar will be recounted
   if(counted_bars>0) counted_bars--;
   limit=Bars-counted_bars;
//---- macd counted in the 1-st additional buffer
   for(int i=0; i<limit; i++)
      ExtMABuffer[i]=iMA(NULL,0,5,0,MODE_SMA,PRICE_MEDIAN,i)-
iMA(NULL,0,34,0,MODE_SMA,PRICE_MEDIAN,i);
//---- dispatch values between 2 buffers
   bool up=true;
   for(i=limit-1; i>=0; i--)
```

```
     {
      current=ExtMABuffer[i];
      prev=ExtMABuffer[i+1];
      if(current>prev) up=true;
      if(current<prev) up=false;
      if(!up)
        {
         ExtRedBuffer[i]=current;
         ExtGreenBuffer[i]=0.0;
        }
      else
        {
         ExtGreenBuffer[i]=current;
         ExtRedBuffer[i]=0.0;
        }
     }
//---- done
   return(0);
  }
//+------------------------------------------------------------------+
```

```
<expert>
  type=EXPERT_ADVISOR
</expert>
#header#
#property copyright "#copyright#"
#property link      "#link#"

#extern_variables#
//+------------------------------------------------------------------+
//| expert initialization function                                   |
//+------------------------------------------------------------------+
int init()
  {
//----

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

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

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

```
<expert>
type=INDICATOR_ADVISOR
</expert>
#header#
#property copyright "#copyright#"
#property link      "#link#"
```

```
#indicator_properties#
#extern_variables#
#mapping_buffers#
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int init()
  {
   #buffers_used#
//---- indicators
   #indicators_init#
//----
   return(0);
  }
//+------------------------------------------------------------------+
//| Custor indicator deinitialization function                       |
//+------------------------------------------------------------------+
int deinit()
  {
//----

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

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

```
<expert>
type=INDICATOR_ADVISOR
separate_window=1
used_buffers=2
<param>
type=int
name=FastEMA
value=12
</param>
<param>
type=int
name=SlowEMA
value=26
</param>
<param>
type=int
name=SignalSMA
value=9
</param>
<ind>
color=Silver
type=DRAW_HISTOGRAM
</ind>
<ind>
color=Red
</ind>
</expert>
#header#
#property copyright "#copyright#"
```

```
#property link      "#link#"

#indicator_properties#
#extern_variables#
#mapping_buffers#
//---- indicator buffers
double ExtSilverBuffer[];
double ExtRedBuffer[];

//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int init()
  {
   #buffers_used#;
//---- drawing settings
   #indicators_init#
//----
   SetIndexDrawBegin(1,SignalSMA);
   IndicatorDigits(5);
//---- indicator buffers mapping
   SetIndexBuffer(0, ExtSilverBuffer);
   SetIndexBuffer(1, ExtRedBuffer);
//---- name for DataWindow and indicator subwindow label
   IndicatorShortName("MACD("+FastEMA+","+SlowEMA+","+SignalSMA+")");
//---- initialization done
   return(0);
  }
//+------------------------------------------------------------------+
//| Moving Averages Convergence/Divergence                           |
//+------------------------------------------------------------------+
int start()
  {
   int limit;
   int counted_bars=IndicatorCounted();
//---- check for possible errors
   if(counted_bars<0) return(-1);
//---- last counted bar will be recounted
   if(counted_bars>0) counted_bars--;
   limit=Bars-counted_bars;
//---- macd counted in the 1-st buffer
   for(int i=0; i<limit; i++)
      ExtSilverBuffer[i]=iMA(NULL,0,FastEMA,0,MODE_EMA,PRICE_CLOSE,i)-
iMA(NULL,0,SlowEMA,0,MODE_EMA,PRICE_CLOSE,i);
//---- signal line counted in the 2-nd buffer
   for(i=0; i<limit; i++)
      ExtRedBuffer[i]=iMAOnArray(ExtSilverBuffer,Bars,SignalSMA,0,MODE_SMA,i);
//---- done
   return(0);
  }
//+------------------------------------------------------------------+
```

```
<expert>
type=INDICATOR_ADVISOR
separate_window=1
used_buffers=3
<param>
name=FastEMA
type=int
value=12
</param>
<param>
name=SlowEMA
type=int
value=26
</param>
<param>
```

```
name=SignalSMA
type=int
value=9
</param>
<ind>
type=DRAW_HISTOGRAM
color=Silver
</ind>
</expert>
#header#
#property copyright "#copyright#"
#property link      "#link#"

#indicator_properties#
#extern_variables#
#mapping_buffers#
//---- indicator buffers
double ExtSilverBuffer[];
double ExtMABuffer[];
double ExtBuffer[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int init()
  {
   #buffers_used#;
//---- drawing settings
   #indicators_init#
//----
   SetIndexDrawBegin(0,SignalSMA);
   IndicatorDigits(6);
//---- indicator buffers mapping
   SetIndexBuffer(0,ExtSilverBuffer);
   SetIndexBuffer(1,ExtMABuffer);
   SetIndexBuffer(2,ExtBuffer);
//---- name for DataWindow and indicator subwindow label
   IndicatorShortName("OsMA("+FastEMA+","+SlowEMA+","+SignalSMA+")");
//---- initialization done
   return(0);
  }
//+------------------------------------------------------------------+
//| Moving Average of Oscillator                                     |
//+------------------------------------------------------------------+
int start()
  {
   int limit;
   int counted_bars=IndicatorCounted();
//---- check for possible errors
   if(counted_bars<0) return(-1);
//---- last counted bar will be recounted
   if(counted_bars>0) counted_bars--;
   limit=Bars-counted_bars;
//---- macd counted in the 1-st additional buffer
   for(int i=0; i<limit; i++)
      ExtMABuffer[i]=iMA(NULL,0,FastEMA,0,MODE_EMA,PRICE_CLOSE,i)-
iMA(NULL,0,SlowEMA,0,MODE_EMA,PRICE_CLOSE,i);
//---- signal line counted in the 2-nd additional buffer
   for(i=0; i<limit; i++)
      ExtBuffer[i]=iMAOnArray(ExtMABuffer,Bars,SignalSMA,0,MODE_SMA,i);
//---- main loop
   for(i=0; i<limit; i++)
      ExtSilverBuffer[i]=ExtMABuffer[i]-ExtBuffer[i];
//---- done
   return(0);
  }
//+------------------------------------------------------------------+
<expert>
type=SCRIPT_ADVISOR
</expert>
#header#
#property copyright "#copyright#"
#property link      "#link#"

#extern_variables#
//+------------------------------------------------------------------+
//| script program start function                                    |
//+------------------------------------------------------------------+
int start()
  {
//----

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

Library

```
//+------------------------------------------------------------------+
//|                                                       stdlib.mq4 |
//|                      Copyright © 2004, MetaQuotes Software Corp. |
//|                                       http://www.metaquotes.net/ |
//+------------------------------------------------------------------+
#property copyright "Copyright © 2004, MetaQuotes Software Corp."
#property link      "http://www.metaquotes.net/"
#property library
/*
int init()
  {
   Print("Init function defined as feature sample");
  }
int deinit()
  {
   Print("Deinit function defined as feature sample too");
  }
*/
//+------------------------------------------------------------------+
//| return error description                                         |
//+------------------------------------------------------------------+
string ErrorDescription(int error_code)
  {
   string error_string;
//----
   switch(error_code)
     {
      //---- codes returned from trade server
      case 0:
      case 1:   error_string="no error";
break;
      case 2:   error_string="common error";
break;
      case 3:   error_string="invalid trade parameters";
break;
      case 4:   error_string="trade server is busy";
break;
      case 5:   error_string="old version of the client terminal";
break;
      case 6:   error_string="no connection with trade server";
break;
      case 7:   error_string="not enough rights";
break;
      case 8:   error_string="too frequent requests";
```

```
break;
      case 9:   error_string="malfunctional trade operation";
break;
      case 64:  error_string="account disabled";
break;
      case 65:  error_string="invalid account";
break;
      case 128: error_string="trade timeout";
break;
      case 129: error_string="invalid price";
break;
      case 130: error_string="invalid stops";
break;
      case 131: error_string="invalid trade volume";
break;
      case 132: error_string="market is closed";
break;
      case 133: error_string="trade is disabled";
break;
      case 134: error_string="not enough money";
break;
      case 135: error_string="price changed";
break;
      case 136: error_string="off quotes";
break;
      case 137: error_string="broker is busy";
break;
      case 138: error_string="requote";
break;
      case 139: error_string="order is locked";
break;
      case 140: error_string="long positions only allowed";
break;
      case 141: error_string="too many requests";
break;
      case 145: error_string="modification denied because order too close to market";
break;
      case 146: error_string="trade context is busy";
break;
      //---- mql4 errors
      case 4000: error_string="no error";
break;
      case 4001: error_string="wrong function pointer";
break;
      case 4002: error_string="array index is out of range";
break;
      case 4003: error_string="no memory for function call stack";
break;
      case 4004: error_string="recursive stack overflow";
break;
      case 4005: error_string="not enough stack for parameter";
break;
      case 4006: error_string="no memory for parameter string";
break;
      case 4007: error_string="no memory for temp string";
break;
      case 4008: error_string="not initialized string";
break;
      case 4009: error_string="not initialized string in array";
break;
      case 4010: error_string="no memory for array\' string";
break;
      case 4011: error_string="too long string";
break;
      case 4012: error_string="remainder from zero divide";
break;
      case 4013: error_string="zero divide";
```

```
break;
      case 4014: error_string="unknown command";
break;
      case 4015: error_string="wrong jump (never generated error)";
break;
      case 4016: error_string="not initialized array";
break;
      case 4017: error_string="dll calls are not allowed";
break;
      case 4018: error_string="cannot load library";
break;
      case 4019: error_string="cannot call function";
break;
      case 4020: error_string="expert function calls are not allowed";
break;
      case 4021: error_string="not enough memory for temp string returned from
function"; break;
      case 4022: error_string="system is busy (never generated error)";
break;
      case 4050: error_string="invalid function parameters count";
break;
      case 4051: error_string="invalid function parameter value";
break;
      case 4052: error_string="string function internal error";
break;
      case 4053: error_string="some array error";
break;
      case 4054: error_string="incorrect series array using";
break;
      case 4055: error_string="custom indicator error";
break;
      case 4056: error_string="arrays are incompatible";
break;
      case 4057: error_string="global variables processing error";
break;
      case 4058: error_string="global variable not found";
break;
      case 4059: error_string="function is not allowed in testing mode";
break;
      case 4060: error_string="function is not confirmed";
break;
      case 4061: error_string="send mail error";
break;
      case 4062: error_string="string parameter expected";
break;
      case 4063: error_string="integer parameter expected";
break;
      case 4064: error_string="double parameter expected";
break;
      case 4065: error_string="array as parameter expected";
break;
      case 4066: error_string="requested history data in update state";
break;
      case 4099: error_string="end of file";
break;
      case 4100: error_string="some file error";
break;
      case 4101: error_string="wrong file name";
break;
      case 4102: error_string="too many opened files";
break;
      case 4103: error_string="cannot open file";
break;
      case 4104: error_string="incompatible access to a file";
break;
      case 4105: error_string="no order selected";
break;
```

```
      case 4106: error_string="unknown symbol";
break;
      case 4107: error_string="invalid price parameter for trade function";
break;
      case 4108: error_string="invalid ticket";
break;
      case 4109: error_string="trade is not allowed";
break;
      case 4110: error_string="longs are not allowed";
break;
      case 4111: error_string="shorts are not allowed";
break;
      case 4200: error_string="object is already exist";
break;
      case 4201: error_string="unknown object property";
break;
      case 4202: error_string="object is not exist";
break;
      case 4203: error_string="unknown object type";
break;
      case 4204: error_string="no object name";
break;
      case 4205: error_string="object coordinates error";
break;
      case 4206: error_string="no specified subwindow";
break;
      default:   error_string="unknown error";
     }
//----
   return(error_string);
  }
//+------------------------------------------------------------------+
//| convert red, green and blue values to color                      |
//+------------------------------------------------------------------+
int RGB(int red_value,int green_value,int blue_value)
  {
//---- check parameters
   if(red_value<0)     red_value=0;
   if(red_value>255)   red_value=255;
   if(green_value<0)   green_value=0;
   if(green_value>255) green_value=255;
   if(blue_value<0)    blue_value=0;
   if(blue_value>255)  blue_value=255;
//----
   green_value<<=8;
   blue_value<<=16;
   return(red_value+green_value+blue_value);
  }
//+------------------------------------------------------------------+
//| right comparison of 2 doubles                                    |
//+------------------------------------------------------------------+
bool CompareDoubles(double number1,double number2)
  {
   if(NormalizeDouble(number1-number2,8)==0) return(true);
   else return(false);
  }
//+------------------------------------------------------------------+
//| up to 16 digits after decimal point                              |
//+------------------------------------------------------------------+
string DoubleToStrMorePrecision(double number,int precision)
  {
   double rem,integer,integer2;
   double DecimalArray[17]={ 1.0, 10.0, 100.0, 1000.0, 10000.0, 100000.0, 1000000.0,
10000000.0, 100000000.0,
                             1000000000.0, 10000000000.0, 100000000000.0,
10000000000000.0, 100000000000000.0,
                             1000000000000000.0, 1000000000000000.0,
10000000000000000.0 };
   string intstring,remstring,retstring;
   bool   isnegative=false;
   int    rem2;
//----
   if(precision<0)  precision=0;
   if(precision>16) precision=16;
//----
   double p=DecimalArray[precision];
   if(number<0.0) { isnegative=true; number=-number; }
   integer=MathFloor(number);
   rem=MathRound((number-integer)*p);
   remstring="";
   for(int i=0; i<precision; i++)
     {
      integer2=MathFloor(rem/10);
      rem2=NormalizeDouble(rem-integer2*10,0);
      remstring=rem2+remstring;
      rem=integer2;
     }
//----
   intstring=DoubleToStr(integer,0);
   if(isnegative) retstring="-"+intstring;
   else           retstring=intstring;
   if(precision>0) retstring=retstring+"."+remstring;
   return(retstring);
  }
//+------------------------------------------------------------------+
//| convert integer to string contained input's hexadecimal notation |
//+------------------------------------------------------------------+
string IntegerToHexString(int integer_number)
  {
   string hex_string="00000000";
   int    value, shift=28;
//   Print("Parameter for IntegerHexToString is ",integer_number);
//----
   for(int i=0; i<8; i++)
     {
      value=(integer_number>>shift)&0x0F;
      if(value<10) hex_string=StringSetChar(hex_string, i, value+'0');
      else         hex_string=StringSetChar(hex_string, i, (value-10)+'A');
      shift-=4;
     }
//----
   return(hex_string);
  }
```

Indicators

```
//+------------------------------------------------------------------+
//|                                                          CCI.mq4 |
//|                      Copyright © 2004, MetaQuotes Software Corp. |
//|                                       http://www.metaquotes.net/ |
//+------------------------------------------------------------------+
#property copyright "Copyright © 2004, MetaQuotes Software Corp."
#property link      "http://www.metaquotes.net/"

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_color1 LightSeaGreen
//---- input parameters
extern int CCIPeriod=14;
//---- buffers
double CCIBuffer[];
double RelBuffer[];
double DevBuffer[];
double MovBuffer[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int init()
  {
   string short_name;
//---- 3 additional buffers are used for counting.
   IndicatorBuffers(4);
   SetIndexBuffer(1, RelBuffer);
   SetIndexBuffer(2, DevBuffer);
   SetIndexBuffer(3, MovBuffer);
//---- indicator lines
   SetIndexStyle(0,DRAW_LINE);
   SetIndexBuffer(0,CCIBuffer);
//---- name for DataWindow and indicator subwindow label
   short_name="CCI("+CCIPeriod+")";
   IndicatorShortName(short_name);
   SetIndexLabel(0,short_name);
//----
   SetIndexDrawBegin(0,CCIPeriod);
//----
   return(0);
  }
//+------------------------------------------------------------------+
//| Commodity Channel Index                                          |
//+------------------------------------------------------------------+
int start()
  {
   int    i,k,counted_bars=IndicatorCounted();
   double price,sum,mul;
   if(Bars<=CCIPeriod) return(0);
//---- initial zero
   if(counted_bars<1)
     {
      for(i=1;i<=CCIPeriod;i++) CCIBuffer[Bars-i]=0.0;
      for(i=1;i<=CCIPeriod;i++) DevBuffer[Bars-i]=0.0;
      for(i=1;i<=CCIPeriod;i++) MovBuffer[Bars-i]=0.0;
     }
//---- last counted bar will be recounted
   int limit=Bars-counted_bars;
   if(counted_bars>0) limit++;
//---- moving average
   for(i=0; i<limit; i++)
      MovBuffer[i]=iMA(NULL,0,CCIPeriod,0,MODE_SMA,PRICE_TYPICAL,i);
//---- standard deviations
   i=Bars-CCIPeriod+1;
   if(counted_bars>CCIPeriod-1) i=Bars-counted_bars-1;
   mul=0.015/CCIPeriod;
   while(i>=0)
     {
      sum=0.0;
      k=i+CCIPeriod-1;
      while(k>=i)
       {
         price=(High[k]+Low[k]+Close[k])/3;
         sum+=MathAbs(price-MovBuffer[i]);
         k--;
       }
      DevBuffer[i]=sum*mul;
      i--;
     }
   i=Bars-CCIPeriod+1;
   if(counted_bars>CCIPeriod-1) i=Bars-counted_bars-1;
   while(i>=0)
     {
      price=(High[i]+Low[i]+Close[i])/3;
      RelBuffer[i]=price-MovBuffer[i];
      i--;
     }
//---- cci counting
   i=Bars-CCIPeriod+1;
   if(counted_bars>CCIPeriod-1) i=Bars-counted_bars-1;
   while(i>=0)
     {
      if(DevBuffer[i]==0.0) CCIBuffer[i]=0.0;
      else CCIBuffer[i]=RelBuffer[i]/DevBuffer[i];
      i--;
     }
//----
   return(0);
  }
//+------------------------------------------------------------------+
```

```
//+------------------------------------------------------------------+
//|                                                     Ichimoku.mq4 |
//|                      Copyright © 2004, MetaQuotes Software Corp. |
//|                                       http://www.metaquotes.net/ |
//+------------------------------------------------------------------+
#property copyright "Copyright © 2004, MetaQuotes Software Corp."
#property link      "http://www.metaquotes.net/"

#property indicator_chart_window
#property indicator_buffers 7
#property indicator_color1 Red
#property indicator_color2 Blue
#property indicator_color3 SandyBrown
#property indicator_color4 Thistle
#property indicator_color5 Lime
#property indicator_color6 SandyBrown
#property indicator_color7 Thistle
//---- input parameters
extern int Tenkan=9;
extern int Kijun=26;
extern int Senkou=52;
//---- buffers
double Tenkan_Buffer[];
double Kijun_Buffer[];
double SpanA_Buffer[];
double SpanB_Buffer[];
double Chinkou_Buffer[];
double SpanA2_Buffer[];
double SpanB2_Buffer[];
//----
int a_begin;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int init()
  {
//----
   SetIndexStyle(0,DRAW_LINE);
   SetIndexBuffer(0,Tenkan_Buffer);
   SetIndexDrawBegin(0,Tenkan-1);
   SetIndexLabel(0,"Tenkan Sen");
//----
   SetIndexStyle(1,DRAW_LINE);
   SetIndexBuffer(1,Kijun_Buffer);
   SetIndexDrawBegin(1,Kijun-1);
   SetIndexLabel(1,"Kijun Sen");
//----
   a_begin=Kijun; if(a_begin<Tenkan) a_begin=Tenkan;
   SetIndexStyle(2,DRAW_HISTOGRAM,STYLE_DOT);
   SetIndexBuffer(2,SpanA_Buffer);
   SetIndexDrawBegin(2,Kijun+a_begin-1);
   SetIndexShift(2,Kijun);
   SetIndexLabel(2,NULL);
   SetIndexStyle(5,DRAW_LINE,STYLE_DOT);
   SetIndexBuffer(5,SpanA2_Buffer);
   SetIndexDrawBegin(5,Kijun+a_begin-1);
   SetIndexShift(5,Kijun);
   SetIndexLabel(5,"Senkou Span A");
//----
   SetIndexStyle(3,DRAW_HISTOGRAM,STYLE_DOT);
   SetIndexBuffer(3,SpanB_Buffer);
   SetIndexDrawBegin(3,Kijun+Senkou-1);
   SetIndexShift(3,Kijun);
   SetIndexLabel(3,NULL);
   SetIndexStyle(6,DRAW_LINE,STYLE_DOT);
   SetIndexBuffer(6,SpanB2_Buffer);
   SetIndexDrawBegin(6,Kijun+Senkou-1);
   SetIndexShift(6,Kijun);
   SetIndexLabel(6,"Senkou Span B");
//----
   SetIndexStyle(4,DRAW_LINE);
   SetIndexBuffer(4,Chinkou_Buffer);
   SetIndexShift(4,-Kijun);
   SetIndexLabel(4,"Chinkou Span");
//----
   return(0);
  }
//+------------------------------------------------------------------+
//| Ichimoku Kinko Hyo                                               |
//+------------------------------------------------------------------+
int start()
  {
   int    i,k;
   int    counted_bars=IndicatorCounted();
   double high,low,price;
//----
   if(Bars<=Tenkan || Bars<=Kijun || Bars<=Senkou) return(0);
//---- initial zero
   if(counted_bars<1)
     {
      for(i=1;i<=Tenkan;i++)    Tenkan_Buffer[Bars-i]=0;
      for(i=1;i<=Kijun;i++)     Kijun_Buffer[Bars-i]=0;
      for(i=1;i<=a_begin;i++) { SpanA_Buffer[Bars-i]=0; SpanA2_Buffer[Bars-i]=0; }
      for(i=1;i<=Senkou;i++)  { SpanB_Buffer[Bars-i]=0; SpanB2_Buffer[Bars-i]=0; }
     }
//---- Tenkan Sen
   i=Bars-Tenkan;
   if(counted_bars>Tenkan) i=Bars-counted_bars-1;
   while(i>=0)
     {
      high=High[i]; low=Low[i]; k=i-1+Tenkan;
      while(k>=i)
        {
         price=High[k];
         if(high<price) high=price;
         price=Low[k];
         if(low>price)  low=price;
         k--;
        }
      Tenkan_Buffer[i]=(high+low)/2;
      i--;
     }
//---- Kijun Sen
   i=Bars-Kijun;
   if(counted_bars>Kijun) i=Bars-counted_bars-1;
   while(i>=0)
     {
      high=High[i]; low=Low[i]; k=i-1+Kijun;
      while(k>=i)
        {
         price=High[k];
         if(high<price) high=price;
         price=Low[k];
         if(low>price)  low=price;
         k--;
        }
      Kijun_Buffer[i]=(high+low)/2;
      i--;
     }
//---- Senkou Span A
   i=Bars-a_begin+1;
   if(counted_bars>a_begin-1) i=Bars-counted_bars-1;
   while(i>=0)
     {
      price=(Kijun_Buffer[i]+Tenkan_Buffer[i])/2;
      SpanA_Buffer[i]=price;
      SpanA2_Buffer[i]=price;
      i--;
     }
//---- Senkou Span B
   i=Bars-Senkou;
   if(counted_bars>Senkou) i=Bars-counted_bars-1;
   while(i>=0)
     {
      high=High[i]; low=Low[i]; k=i-1+Senkou;
      while(k>=i)
        {
         price=High[k];
         if(high<price) high=price;
         price=Low[k];
         if(low>price)  low=price;
         k--;
        }
      price=(high+low)/2;
      SpanB_Buffer[i]=price;
      SpanB2_Buffer[i]=price;
      i--;
     }
//---- Chinkou Span
   i=Bars-1;
   if(counted_bars>1) i=Bars-counted_bars-1;
   while(i>=0) { Chinkou_Buffer[i]=Close[i]; i--; }
//----
   return(0);
  }
//+------------------------------------------------------------------+
```

```
//+------------------------------------------------------------------+
//|                                                  Custom MACD.mq4 |
//|                      Copyright © 2004, MetaQuotes Software Corp. |
//|                                       http://www.metaquotes.net/ |
//+------------------------------------------------------------------+
#property  copyright "Copyright © 2004, MetaQuotes Software Corp."
#property  link      "http://www.metaquotes.net/"
//---- indicator settings
#property  indicator_separate_window
#property  indicator_buffers 2
#property  indicator_color1  Silver
#property  indicator_color2  Red
//---- indicator parameters
extern int FastEMA=12;
extern int SlowEMA=26;
extern int SignalSMA=9;
//---- indicator buffers
```

```
double     ind_buffer1[];
double     ind_buffer2[];

//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int init()
  {
//---- drawing settings
   SetIndexStyle(0,DRAW_HISTOGRAM,STYLE_SOLID,3);
   SetIndexDrawBegin(1,SignalSMA);
   IndicatorDigits(MarketInfo(Symbol(),MODE_DIGITS)+1);
//---- indicator buffers mapping
   if(!SetIndexBuffer(0,ind_buffer1) && !SetIndexBuffer(1,ind_buffer2))
      Print("cannot set indicator buffers!");
//---- name for DataWindow and indicator subwindow label
   IndicatorShortName("MACD("+FastEMA+","+SlowEMA+","+SignalSMA+")");
   SetIndexLabel(0,"MACD");
   SetIndexLabel(1,"Signal");
//---- initialization done
   return(0);
  }
//+------------------------------------------------------------------+
//| Moving Averages Convergence/Divergence                           |
//+------------------------------------------------------------------+
int start()
  {
   int limit;
   int counted_bars=IndicatorCounted();
//---- check for possible errors
   if(counted_bars<0) return(-1);
//---- last counted bar will be recounted
   if(counted_bars>0) counted_bars--;
   limit=Bars-counted_bars;
//---- macd counted in the 1-st buffer
   for(int i=0; i<limit; i++)
      ind_buffer1[i]=iMA(NULL,0,FastEMA,0,MODE_EMA,PRICE_CLOSE,i)-
iMA(NULL,0,SlowEMA,0,MODE_EMA,PRICE_CLOSE,i);
//---- signal line counted in the 2-nd buffer
   for(i=0; i<limit; i++)
      ind_buffer2[i]=iMAOnArray(ind_buffer1,Bars,SignalSMA,0,MODE_SMA,i);
//---- done
   return(0);
  }
```

```
//+------------------------------------------------------------------+
//|                                                     Momentum.mq4 |
//|                      Copyright © 2004, MetaQuotes Software Corp. |
//|                                       http://www.metaquotes.net/ |
//+------------------------------------------------------------------+
#property copyright "Copyright © 2004, MetaQuotes Software Corp."
#property link      "http://www.metaquotes.net/"

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_color1 DodgerBlue
//---- input parameters
extern int MomPeriod=14;
//---- buffers
double MomBuffer[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int init()
  {
   string short_name;
//---- indicator line
```

```
   SetIndexStyle(0,DRAW_LINE);
   SetIndexBuffer(0,MomBuffer);
//---- name for DataWindow and indicator subwindow label
   short_name="Mom("+MomPeriod+")";
   IndicatorShortName(short_name);
   SetIndexLabel(0,short_name);
//----
   SetIndexDrawBegin(0,MomPeriod);
//----
   return(0);
  }
//+------------------------------------------------------------------+
//| Momentum                                                         |
//+------------------------------------------------------------------+
int start()
  {
   int i,counted_bars=IndicatorCounted();
//----
   if(Bars<=MomPeriod) return(0);
//---- initial zero
   if(counted_bars<1)
      for(i=1;i<=MomPeriod;i++) MomBuffer[Bars-i]=0.0;
//----
   i=Bars-MomPeriod-1;
   if(counted_bars>=MomPeriod) i=Bars-counted_bars-1;
   while(i>=0)
     {
      MomBuffer[i]=Close[i]*100/Close[i+MomPeriod];
      i--;
     }
   return(0);
  }
//+------------------------------------------------------------------+
```

```
//+------------------------------------------------------------------+
//|                                        Custom Moving Average.mq4 |
//|                      Copyright © 2004, MetaQuotes Software Corp. |
//|                                       http://www.metaquotes.net/ |
//+------------------------------------------------------------------+
#property copyright "Copyright © 2004, MetaQuotes Software Corp."
#property link      "http://www.metaquotes.net/"

#property indicator_chart_window
#property indicator_buffers 1
#property indicator_color1 Red
//---- indicator parameters
extern int MA_Period=13;
extern int MA_Shift=0;
extern int MA_Method=0;
//---- indicator buffers
double ExtMapBuffer[];
//----
int ExtCountedBars=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int init()
  {
   int    draw_begin;
   string short_name;
//---- drawing settings
   SetIndexStyle(0,DRAW_LINE);
   SetIndexShift(0,MA_Shift);
   IndicatorDigits(MarketInfo(Symbol(),MODE_DIGITS));
   if(MA_Period<2) MA_Period=13;
   draw_begin=MA_Period-1;
//---- indicator short name
```

```
   switch(MA_Method)
     {
      case 1 : short_name="EMA(";  draw_begin=0; break;
      case 2 : short_name="SMMA("; break;
      case 3 : short_name="LWMA("; break;
      default :
         MA_Method=0;
         short_name="SMA(";
     }
   IndicatorShortName(short_name+MA_Period+")");
   SetIndexDrawBegin(0,draw_begin);
//---- indicator buffers mapping
   SetIndexBuffer(0,ExtMapBuffer);
//---- initialization done
   return(0);
  }
//+------------------------------------------------------------------+
//|                                                                  |
//+------------------------------------------------------------------+
int start()
  {
   if(Bars<=MA_Period) return(0);
   ExtCountedBars=IndicatorCounted();
//---- check for possible errors
   if (ExtCountedBars<0) return(-1);
//---- last counted bar will be recounted
   if (ExtCountedBars>0) ExtCountedBars--;
//----
   switch(MA_Method)
     {
      case 0 : sma();  break;
      case 1 : ema();  break;
      case 2 : smma(); break;
      case 3 : lwma();
     }
//---- done
   return(0);
  }
//+------------------------------------------------------------------+
//| Simple Moving Average                                            |
//+------------------------------------------------------------------+
void sma()
  {
   double sum=0;
   int    i,pos=Bars-ExtCountedBars-1;
//---- initial accumulation
   if(pos<MA_Period) pos=MA_Period;
   for(i=1;i<MA_Period;i++,pos--)
      sum+=Close[pos];
//---- main calculation loop
   while(pos>=0)
     {
      sum+=Close[pos];
      ExtMapBuffer[pos]=sum/MA_Period;
         sum-=Close[pos+MA_Period-1];
         pos--;
     }
//---- zero initial bars
   if(ExtCountedBars<1)
      for(i=1;i<MA_Period;i++) ExtMapBuffer[Bars-i]=0;
  }
//+------------------------------------------------------------------+
//| Exponential Moving Average                                       |
//+------------------------------------------------------------------+
void ema()
  {
   double pr=2.0/(MA_Period+1);
   int    pos=Bars-2;
   if(ExtCountedBars>2) pos=Bars-ExtCountedBars-1;
//---- main calculation loop
   while(pos>=0)
     {
      if(pos==Bars-2) ExtMapBuffer[pos+1]=Close[pos+1];
      ExtMapBuffer[pos]=Close[pos]*pr+ExtMapBuffer[pos+1]*(1-pr);
         pos--;
     }
  }
//+------------------------------------------------------------------+
//| Smoothed Moving Average                                          |
//+------------------------------------------------------------------+
void smma()
  {
   double sum=0;
   int    i,k,pos=Bars-ExtCountedBars+1;
//---- main calculation loop
   pos=Bars-MA_Period;
   if(pos>Bars-ExtCountedBars) pos=Bars-ExtCountedBars;
   while(pos>=0)
     {
      if(pos==Bars-MA_Period)
        {
         //---- initial accumulation
         for(i=0,k=pos;i<MA_Period;i++,k++)
           {
            sum+=Close[k];
            //---- zero initial bars
            ExtMapBuffer[k]=0;
           }
        }
      else sum=ExtMapBuffer[pos+1]*(MA_Period-1)+Close[pos];
      ExtMapBuffer[pos]=sum/MA_Period;
         pos--;
     }
  }
//+------------------------------------------------------------------+
//| Linear Weighted Moving Average                                   |
//+------------------------------------------------------------------+
void lwma()
  {
   double sum=0.0,lsum=0.0;
   double price;
   int    i,weight=0,pos=Bars-ExtCountedBars-1;
//---- initial accumulation
   if(pos<MA_Period) pos=MA_Period;
   for(i=1;i<=MA_Period;i++,pos--)
     {
      price=Close[pos];
      sum+=price*i;
      lsum+=price;
      weight+=i;
     }
//---- main calculation loop
   pos++;
   i=pos+MA_Period;
   while(pos>=0)
     {
      ExtMapBuffer[pos]=sum/weight;
      if(pos==0) break;
      pos--;
      i--;
      price=Close[pos];
      sum=sum-lsum+price*MA_Period;
      lsum-=Close[i];
      lsum+=price;
     }
//---- zero initial bars
   if(ExtCountedBars<1)
      for(i=1;i<MA_Period;i++) ExtMapBuffer[Bars-i]=0;
  }
//+------------------------------------------------------------------+
```

```
//+------------------------------------------------------------------+
//|                                                         OsMA.mq4 |
//|                      Copyright © 2004, MetaQuotes Software Corp. |
//|                                       http://www.metaquotes.net/ |
//+------------------------------------------------------------------+
#property  copyright "Copyright © 2004, MetaQuotes Software Corp."
#property  link      "http://www.metaquotes.net/"
//---- indicator settings
#property  indicator_separate_window
#property  indicator_buffers 1
#property  indicator_color1  Silver
//---- indicator parameters
extern int FastEMA=12;
extern int SlowEMA=26;
extern int SignalSMA=9;
//---- indicator buffers
double     ind_buffer1[];
double     ind_buffer2[];
double     ind_buffer3[];

//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int init()
  {
//---- 2 additional buffers are used for counting.
   IndicatorBuffers(3);
//---- drawing settings
   SetIndexStyle(0,DRAW_HISTOGRAM,STYLE_SOLID,3);
   SetIndexDrawBegin(0,SignalSMA);
   IndicatorDigits(MarketInfo(Symbol(),MODE_DIGITS)+2);
//---- 3 indicator buffers mapping
   if(!SetIndexBuffer(0,ind_buffer1) &&
      !SetIndexBuffer(1,ind_buffer2) &&
      !SetIndexBuffer(2,ind_buffer3))
      Print("cannot set indicator buffers!");
//---- name for DataWindow and indicator subwindow label
   IndicatorShortName("OsMA("+FastEMA+","+SlowEMA+","+SignalSMA+")");
//---- initialization done
   return(0);
  }
//+------------------------------------------------------------------+
//| Moving Average of Oscillator                                     |
//+------------------------------------------------------------------+
int start()
  {
   int limit;
   int counted_bars=IndicatorCounted();
//---- check for possible errors
   if(counted_bars<0) return(-1);
//---- last counted bar will be recounted
   if(counted_bars>0) counted_bars--;
   limit=Bars-counted_bars;
//---- macd counted in the 1-st additional buffer
   for(int i=0; i<limit; i++)
      ind_buffer2[i]=iMA(NULL,0,FastEMA,0,MODE_EMA,PRICE_CLOSE,i)-
iMA(NULL,0,SlowEMA,0,MODE_EMA,PRICE_CLOSE,i);
//---- signal line counted in the 2-nd additional buffer
   for(i=0; i<limit; i++)
      ind_buffer3[i]=iMAOnArray(ind_buffer2,Bars,SignalSMA,0,MODE_SMA,i);
//---- main loop
   for(i=0; i<limit; i++)
      ind_buffer1[i]=ind_buffer2[i]-ind_buffer3[i];
//---- done
   return(0);
  }
//+------------------------------------------------------------------+
```

```
//+------------------------------------------------------------------+
//|                                                    Parabolic.mq4 |
//|                      Copyright © 2004, MetaQuotes Software Corp. |
//|                                       http://www.metaquotes.net/ |
//+------------------------------------------------------------------+
#property copyright "Copyright © 2004, MetaQuotes Software Corp."
#property link      "http://www.metaquotes.net/"

#property indicator_chart_window
#property indicator_buffers 1
#property indicator_color1 Lime
//---- input parameters
extern double    Step=0.02;
extern double    Maximum=0.2;
//---- buffers
double SarBuffer[];
//----
int    save_lastreverse;
bool   save_dirlong;
double save_start;
double save_last_high;
double save_last_low;
double save_ep;
double save_sar;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int init()
  {
//---- indicators
   IndicatorDigits(Digits);
   SetIndexStyle(0,DRAW_ARROW);
   SetIndexArrow(0,159);
   SetIndexBuffer(0,SarBuffer);
//----
   return(0);
  }
//+------------------------------------------------------------------+
//|                                                                  |
//+------------------------------------------------------------------+
void SaveLastReverse(int last,int dir,double start,double low,double high,double
ep,double sar)
  {
   save_lastreverse=last;
   save_dirlong=dir;
   save_start=start;
   save_last_low=low;
   save_last_high=high;
   save_ep=ep;
   save_sar=sar;
  }
//+------------------------------------------------------------------+
//| Parabolic Sell And Reverse system                                |
//+------------------------------------------------------------------+
int start()
  {
   static bool first=true;
   bool   dirlong;
   double start,last_high,last_low;
   double ep,sar,price_low,price_high,price;
   int    i,counted_bars=IndicatorCounted();
//----
   if(Bars<3) return(0);
//---- initial settings
   i=Bars-2;
   if(counted_bars==0 || first)
     {
      first=false;
      dirlong=true;
      start=Step;
      last_high=-10000000.0;
      last_low=10000000.0;
      while(i>0)
        {
         save_lastreverse=i;
         price_low=Low[i];
         if(last_low>price_low)   last_low=price_low;
         price_high=High[i];
         if(last_high<price_high) last_high=price_high;
         if(price_high>High[i+1] && price_low>Low[i+1]) break;
         if(price_high<High[i+1] && price_low<Low[i+1]) { dirlong=false; break; }
         i--;
        }
      //---- initial zero
      int k=i;
      while(k<Bars)
        {
         SarBuffer[k]=0.0;
         k++;
        }
      //---- check further
      if(dirlong) { SarBuffer[i]=Low[i+1]; ep=High[i]; }
      else        { SarBuffer[i]=High[i+1]; ep=Low[i]; }
      i--;
     }
    else
     {
      i=save_lastreverse+1;
      start=save_start;
      dirlong=save_dirlong;
      last_high=save_last_high;
      last_low=save_last_low;
      ep=save_ep;
      sar=save_sar;
     }
//----
   while(i>=0)
     {
      price_low=Low[i];
      price_high=High[i];
      //--- check for reverse
      if(dirlong && price_low<SarBuffer[i+1])
        {
         SaveLastReverse(i,true,start,price_low,last_high,ep,sar);
         start=Step; dirlong=false;
         ep=price_low;  last_low=price_low;
         SarBuffer[i]=last_high;
         i--;
         continue;
        }
      if(!dirlong && price_high>SarBuffer[i+1])
        {
         SaveLastReverse(i,false,start,last_low,price_high,ep,sar);
         start=Step; dirlong=true;
         ep=price_high; last_high=price_high;
         SarBuffer[i]=last_low;
         i--;
         continue;
        }
      //---
      price=SarBuffer[i+1];
      sar=price+start*(ep-price);
      if(dirlong)
        {
         if(ep<price_high && (start+Step)<=Maximum) start+=Step;
         if(price_high<High[i+1] && i==Bars-2)  sar=SarBuffer[i+1];

         price=Low[i+1];
         if(sar>price) sar=price;
         price=Low[i+2];
         if(sar>price) sar=price;
         if(sar>price_low)
           {
            SaveLastReverse(i,true,start,price_low,last_high,ep,sar);
            start=Step; dirlong=false; ep=price_low;
            last_low=price_low;
            SarBuffer[i]=last_high;
            i--;
            continue;
           }
         if(ep<price_high) { last_high=price_high; ep=price_high; }
        }
      else
        {
         if(ep>price_low && (start+Step)<=Maximum) start+=Step;
         if(price_low<Low[i+1] && i==Bars-2)  sar=SarBuffer[i+1];

         price=High[i+1];
         if(sar<price) sar=price;
         price=High[i+2];
         if(sar<price) sar=price;
         if(sar<price_high)
           {
            SaveLastReverse(i,false,start,last_low,price_high,ep,sar);
            start=Step; dirlong=true; ep=price_high;
            last_high=price_high;
            SarBuffer[i]=last_low;
            i--;
            continue;
           }
         if(ep>price_low) { last_low=price_low; ep=price_low; }
        }
      SarBuffer[i]=sar;
      i--;
     }
//   sar=SarBuffer[0];
//   price=iSAR(NULL,0,Step,Maximum,0);
//   if(sar!=price) Print("custom=",sar,"   SAR=",price,"   counted=",counted_bars);
//   if(sar==price) Print("custom=",sar,"   SAR=",price,"   counted=",counted_bars);
//----
   return(0);
  }
//+------------------------------------------------------------------+
```

```
//+------------------------------------------------------------------+
//|                                                          RSI.mq4 |
//|                      Copyright © 2004, MetaQuotes Software Corp. |
//|                                       http://www.metaquotes.net/ |
//+------------------------------------------------------------------+
#property copyright "Copyright © 2004, MetaQuotes Software Corp."
#property link      "http://www.metaquotes.net/"

#property indicator_separate_window
```

```
#property indicator_minimum 0
#property indicator_maximum 100
#property indicator_buffers 1
#property indicator_color1 DodgerBlue
//---- input parameters
extern int RSIPeriod=14;
//---- buffers
double RSIBuffer[];
double PosBuffer[];
double NegBuffer[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int init()
  {
   string short_name;
//---- 2 additional buffers are used for counting.
   IndicatorBuffers(3);
   SetIndexBuffer(1,PosBuffer);
   SetIndexBuffer(2,NegBuffer);
//---- indicator line
   SetIndexStyle(0,DRAW_LINE);
   SetIndexBuffer(0,RSIBuffer);
//---- name for DataWindow and indicator subwindow label
   short_name="RSI("+RSIPeriod+")";
   IndicatorShortName(short_name);
   SetIndexLabel(0,short_name);
//----
   SetIndexDrawBegin(0,RSIPeriod);
//----
   return(0);
  }
//+------------------------------------------------------------------+
//| Relative Strength Index                                          |
//+------------------------------------------------------------------+
int start()
  {
   int    i,counted_bars=IndicatorCounted();
   double rel,negative,positive;
//----
   if(Bars<=RSIPeriod) return(0);
//---- initial zero
   if(counted_bars<1)
      for(i=1;i<=RSIPeriod;i++) RSIBuffer[Bars-i]=0.0;
//----
   i=Bars-RSIPeriod-1;
   if(counted_bars>=RSIPeriod) i=Bars-counted_bars-1;
   while(i>=0)
     {
      double sumn=0.0,sump=0.0;
      if(i==Bars-RSIPeriod-1)
        {
         int k=Bars-2;
         //---- initial accumulation
         while(k>=i)
           {
            rel=Close[k]-Close[k+1];
            if(rel>0) sump+=rel;
            else      sumn-=rel;
            k--;
           }
         positive=sump/RSIPeriod;
         negative=sumn/RSIPeriod;
        }
      else
        {
         //---- smoothed moving average
         rel=Close[i]-Close[i+1];
         if(rel>0) sump=rel;
         else      sumn=-rel;
         positive=(PosBuffer[i+1]*(RSIPeriod-1)+sump)/RSIPeriod;
         negative=(NegBuffer[i+1]*(RSIPeriod-1)+sumn)/RSIPeriod;
        }
      PosBuffer[i]=positive;
      NegBuffer[i]=negative;
      if(negative==0.0) RSIBuffer[i]=0.0;
      else RSIBuffer[i]=100.0-100.0/(1+positive/negative);
      i--;
     }
//----
   return(0);
  }
//+------------------------------------------------------------------+
```

```
//+------------------------------------------------------------------+
//|                                                   Stochastic.mq4 |
//|                      Copyright © 2004, MetaQuotes Software Corp. |
//|                                       http://www.metaquotes.net/ |
//+------------------------------------------------------------------+
#property copyright "Copyright © 2004, MetaQuotes Software Corp."
#property link      "http://www.metaquotes.net/"

#property indicator_separate_window
#property indicator_minimum 0
#property indicator_maximum 100
#property indicator_buffers 2
#property indicator_color1 LightSeaGreen
#property indicator_color2 Red
//---- input parameters
extern int KPeriod=5;
extern int DPeriod=3;
extern int Slowing=3;
//---- buffers
double MainBuffer[];
double SignalBuffer[];
double HighesBuffer[];
double LowesBuffer[];
//----
int draw_begin1=0;
int draw_begin2=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int init()
  {
   string short_name;
//---- 2 additional buffers are used for counting.
   IndicatorBuffers(4);
   SetIndexBuffer(2, HighesBuffer);
   SetIndexBuffer(3, LowesBuffer);
//---- indicator lines
   SetIndexStyle(0,DRAW_LINE);
   SetIndexBuffer(0, MainBuffer);
   SetIndexStyle(1,DRAW_LINE);
   SetIndexBuffer(1, SignalBuffer);
//---- name for DataWindow and indicator subwindow label
   short_name="Sto("+KPeriod+","+DPeriod+","+Slowing+")";
   IndicatorShortName(short_name);
   SetIndexLabel(0,short_name);
   SetIndexLabel(1,"Signal");
//----
   draw_begin1=KPeriod+Slowing;
   draw_begin2=draw_begin1+DPeriod;
   SetIndexDrawBegin(0,draw_begin1);
   SetIndexDrawBegin(1,draw_begin2);
//----
   return(0);
  }
//+------------------------------------------------------------------+
//| Stochastic oscillator                                            |
//+------------------------------------------------------------------+
int start()
  {
   int    i,k;
   int    counted_bars=IndicatorCounted();
   double price;
//----
   if(Bars<=draw_begin2) return(0);
//---- initial zero
   if(counted_bars<1)
     {
      for(i=1;i<=draw_begin1;i++) MainBuffer[Bars-i]=0;
      for(i=1;i<=draw_begin2;i++) SignalBuffer[Bars-i]=0;
     }
//---- minimums counting
   i=Bars-KPeriod;
   if(counted_bars>KPeriod) i=Bars-counted_bars-1;
   while(i>=0)
     {
      double min=1000000;
      k=i+KPeriod-1;
      while(k>=i)
        {
         price=Low[k];
         if(min>price) min=price;
         k--;
        }
      LowesBuffer[i]=min;
      i--;
     }
//---- maximums counting
   i=Bars-KPeriod;
   if(counted_bars>KPeriod) i=Bars-counted_bars-1;
   while(i>=0)
     {
      double max=-1000000;
      k=i+KPeriod-1;
      while(k>=i)
        {
         price=High[k];
         if(max<price) max=price;
         k--;
        }
      HighesBuffer[i]=max;
      i--;
     }
//---- %K line
   i=Bars-draw_begin1;
   if(counted_bars>draw_begin1) i=Bars-counted_bars-1;
   while(i>=0)
     {
      double sumlow=0.0;
      double sumhigh=0.0;
      for(k=(i+Slowing-1);k>=i;k--)
        {
         sumlow+=Close[k]-LowesBuffer[k];
         sumhigh+=HighesBuffer[k]-LowesBuffer[k];
        }
      if(sumhigh==0.0) MainBuffer[i]=100.0;
      else MainBuffer[i]=sumlow/sumhigh*100;
      i--;
     }
//---- last counted bar will be recounted
   if(counted_bars>0) counted_bars--;
   int limit=Bars-counted_bars;
//---- signal line is simple movimg average
   for(i=0; i<limit; i++)
      SignalBuffer[i]=iMAOnArray(MainBuffer,Bars,DPeriod,0,MODE_SMA,i);
//----
   return(0);
  }
//+------------------------------------------------------------------+
```

```
//+------------------------------------------------------------------+
//|                                      Custom Moving Average.mq4 |
//|                      Copyright © 2005, MetaQuotes Software Corp. |
//|                                       http://www.metaquotes.net/ |
//+------------------------------------------------------------------+
#property copyright "Copyright © 2005, MetaQuotes Software Corp."
#property link      "http://www.metaquotes.net/"

#property indicator_chart_window
#property indicator_buffers 1
#property indicator_color1 Red
//---- indicator parameters
extern int ExtDepth=12;
extern int ExtDeviation=5;
extern int ExtBackstep=3;
//---- indicator buffers
double ExtMapBuffer[];
double ExtMapBuffer2[];

//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int init()
  {
   IndicatorBuffers(2);
//---- drawing settings
   SetIndexStyle(0,DRAW_SECTION);
//---- indicator buffers mapping
   SetIndexBuffer(0,ExtMapBuffer);
   SetIndexBuffer(1,ExtMapBuffer2);
   SetIndexEmptyValue(0,0.0);
   ArraySetAsSeries(ExtMapBuffer,true);
   ArraySetAsSeries(ExtMapBuffer2,true);
//---- indicator short name
   IndicatorShortName("ZigZag("+ExtDepth+","+ExtDeviation+","+ExtBackstep+")");
//---- initialization done
   return(0);
  }
//+------------------------------------------------------------------+
//|                                                                  |
//+------------------------------------------------------------------+
int start()
  {
   int    shift, back,lasthighpos,lastlowpos;
   double val,res;
   double curlow,curhigh,lasthigh,lastlow;

   for(shift=Bars-ExtDepth; shift>=0; shift--)
     {
      val=Low[Lowest(NULL,0,MODE_LOW,ExtDepth,shift)];
      if(val==lastlow) val=0.0;
      else
        {
         lastlow=val;
         if((Low[shift]-val)>(ExtDeviation*Point)) val=0.0;
         else
           {
            for(back=1; back<=ExtBackstep; back++)
              {
               res=ExtMapBuffer[shift+back];
               if((res!=0)&&(res>val)) ExtMapBuffer[shift+back]=0.0;
              }
           }
        }
      ExtMapBuffer[shift]=val;
      //--- high
      val=High[Highest(NULL,0,MODE_HIGH,ExtDepth,shift)];
      if(val==lasthigh) val=0.0;
      else
        {
         lasthigh=val;
         if((val-High[shift])>(ExtDeviation*Point)) val=0.0;
         else
           {
            for(back=1; back<=ExtBackstep; back++)
              {
               res=ExtMapBuffer2[shift+back];
               if((res!=0)&&(res<val)) ExtMapBuffer2[shift+back]=0.0;
              }
           }
        }
      ExtMapBuffer2[shift]=val;
     }

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

   for(shift=Bars-ExtDepth; shift>=0; shift--)
     {
      curlow=ExtMapBuffer[shift];
      curhigh=ExtMapBuffer2[shift];
      if((curlow==0)&&(curhigh==0)) continue;
      //---
      if(curhigh!=0)
        {
         if(lasthigh>0)
           {
            if(lasthigh<curhigh) ExtMapBuffer2[lasthighpos]=0;
            else ExtMapBuffer2[shift]=0;
           }
         //---
         if(lasthigh<curhigh || lasthigh<0)
           {
            lasthigh=curhigh;
            lasthighpos=shift;
           }
         lastlow=-1;
        }
      //----
      if(curlow!=0)
        {
         if(lastlow>0)
           {
            if(lastlow>curlow) ExtMapBuffer[lastlowpos]=0;
            else ExtMapBuffer[shift]=0;
           }
         //---
         if((curlow<lastlow)||(lastlow<0))
           {
            lastlow=curlow;
            lastlowpos=shift;
           }
         lasthigh=-1;
        }
     }

   for(shift=Bars-1; shift>=0; shift--)
     {
      if(shift>=Bars-ExtDepth) ExtMapBuffer[shift]=0.0;
      else
        {
         res=ExtMapBuffer2[shift];
         if(res!=0.0) ExtMapBuffer[shift]=res;
        }
     }
  }
```

```
//+------------------------------------------------------------------+
//|                                                  Accelerator.mq4 |
//|                      Copyright © 2005, MetaQuotes Software Corp. |
//|                                       http://www.metaquotes.net/ |
//+------------------------------------------------------------------+
#property  copyright "Copyright © 2005, MetaQuotes Software Corp."
#property  link      "http://www.metaquotes.net/"
//---- indicator settings
#property  indicator_separate_window
#property  indicator_buffers 3
#property  indicator_color1  Black
#property  indicator_color2  Green
#property  indicator_color3  Red
//---- indicator buffers
double     ExtBuffer0[];
double     ExtBuffer1[];
double     ExtBuffer2[];
double     ExtBuffer3[];
double     ExtBuffer4[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int init()
  {
//---- 2 additional buffers are used for counting.
   IndicatorBuffers(5);
//---- drawing settings
   SetIndexStyle(0,DRAW_NONE);
   SetIndexStyle(1,DRAW_HISTOGRAM);
   SetIndexStyle(2,DRAW_HISTOGRAM);
   IndicatorDigits(Digits+2);
   SetIndexDrawBegin(0,38);
   SetIndexDrawBegin(1,38);
   SetIndexDrawBegin(2,38);
//---- 4 indicator buffers mapping
   SetIndexBuffer(0,ExtBuffer0);
   SetIndexBuffer(1,ExtBuffer1);
   SetIndexBuffer(2,ExtBuffer2);
   SetIndexBuffer(3,ExtBuffer3);
   SetIndexBuffer(4,ExtBuffer4);
//---- name for DataWindow and indicator subwindow label
   IndicatorShortName("AC");
   SetIndexLabel(1,NULL);
   SetIndexLabel(2,NULL);
//---- initialization done
   return(0);
  }
//+------------------------------------------------------------------+
//| Accelerator/Decelerator Oscillator                               |
//+------------------------------------------------------------------+
int start()
```

```
  {
   int    limit;
   int    counted_bars=IndicatorCounted();
   double prev,current;
   //---- last counted bar will be recounted
   if(counted_bars>0) counted_bars--;
   limit=Bars-counted_bars;
   //---- macd counted in the 1-st additional buffer
   for(int i=0; i<limit; i++)
      ExtBuffer3[i]=iMA(NULL,0,5,0,MODE_SMA,PRICE_MEDIAN,i)-
iMA(NULL,0,34,0,MODE_SMA,PRICE_MEDIAN,i);
   //---- signal line counted in the 2-nd additional buffer
   for(i=0; i<limit; i++)
      ExtBuffer4[i]=iMAOnArray(ExtBuffer3,Bars,5,0,MODE_SMA,i);
   //---- dispatch values between 2 buffers
   bool up=true;
   for(i=limit-1; i>=0; i--)
     {
      current=ExtBuffer3[i]-ExtBuffer4[i];
      prev=ExtBuffer3[i+1]-ExtBuffer4[i+1];
      if(current>prev) up=true;
      if(current<prev) up=false;
      if(!up)
        {
         ExtBuffer2[i]=current;
         ExtBuffer1[i]=0.0;
        }
      else
        {
         ExtBuffer1[i]=current;
         ExtBuffer2[i]=0.0;
        }
       ExtBuffer0[i]=current;
     }
   //---- done
   return(0);
  }
```

```
//+------------------------------------------------------------------+
//|                                                          ADX.mq4 |
//|                      Copyright © 2004, MetaQuotes Software Corp. |
//|                                       http://www.metaquotes.net/ |
//+------------------------------------------------------------------+
#property copyright "Copyright © 2004, MetaQuotes Software Corp."
#property link      "http://www.metaquotes.net/"

#property indicator_separate_window
#property indicator_buffers 3
#property indicator_color1 LightSeaGreen
#property indicator_color2 YellowGreen
#property indicator_color3 Wheat
//---- input parameters
extern int ADXPeriod=14;
//---- buffers
double ADXBuffer[];
double PlusDiBuffer[];
double MinusDiBuffer[];
double PlusSdiBuffer[];
double MinusSdiBuffer[];
double TempBuffer[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int init()
  {
//---- 3 additional buffers are used for counting.
   IndicatorBuffers(6);
```

```
//---- indicator buffers
   SetIndexBuffer(0,ADXBuffer);
   SetIndexBuffer(1,PlusDiBuffer);
   SetIndexBuffer(2,MinusDiBuffer);
   SetIndexBuffer(3,PlusSdiBuffer);
   SetIndexBuffer(4,MinusSdiBuffer);
   SetIndexBuffer(5,TempBuffer);
//---- name for DataWindow and indicator subwindow label
   IndicatorShortName("ADX("+ADXPeriod+")");
   SetIndexLabel(0,"ADX");
   SetIndexLabel(1,"+DI");
   SetIndexLabel(2,"-DI");
//----
   SetIndexDrawBegin(0,ADXPeriod);
   SetIndexDrawBegin(1,ADXPeriod);
   SetIndexDrawBegin(2,ADXPeriod);
//----
   return(0);
  }
//+------------------------------------------------------------------+
//| Average Directional Movement Index                               |
//+------------------------------------------------------------------+
int start()
  {
   double pdm,mdm,tr;
   double price_high,price_low;
   int    starti,i,counted_bars=IndicatorCounted();
//----
   i=Bars-2;
   PlusSdiBuffer[i+1]=0;
   MinusSdiBuffer[i+1]=0;
   if(counted_bars>=i) i=Bars-counted_bars-1;
   starti=i;
//----
   while(i>=0)
     {
      price_low=Low[i];
      price_high=High[i];
      //----
      pdm=price_high-High[i+1];
      mdm=Low[i+1]-price_low;
      if(pdm<0) pdm=0;  // +DM
      if(mdm<0) mdm=0;  // -DM
      if(pdm==mdm) { pdm=0; mdm=0; }
      else if(pdm<mdm) pdm=0;
           else if(mdm<pdm) mdm=0;
      //---- âû÷èñëÿåì èñòèííûé èíòåðâàë
      double num1=MathAbs(price_high-price_low);
      double num2=MathAbs(price_high-Close[i+1]);
      double num3=MathAbs(price_low-Close[i+1]);
      tr=MathMax(num1,num2);
      tr=MathMax(tr,num3);
      //---- counting plus/minus direction
      if(tr==0) { PlusSdiBuffer[i]=0; MinusSdiBuffer[i]=0; }
      else      { PlusSdiBuffer[i]=100.0*pdm/tr; MinusSdiBuffer[i]=100.0*mdm/tr; }
      //----
      i--;
     }
//---- last counted bar will be recounted
   if(counted_bars>0) counted_bars--;
   int limit=Bars-counted_bars;
//---- apply EMA to +DI
   for(i=0; i<=limit; i++)
      PlusDiBuffer[i]=iMAOnArray(PlusSdiBuffer,Bars,ADXPeriod,0,MODE_EMA,i);
//---- apply EMA to -DI
   for(i=0; i<=limit; i++)
      MinusDiBuffer[i]=iMAOnArray(MinusSdiBuffer,Bars,ADXPeriod,0,MODE_EMA,i);
```

```
//---- Directional Movement (DX)
   i=Bars-2;
   TempBuffer[i+1]=0;
   i=starti;
   while(i>=0)
     {
      double div=MathAbs(PlusDiBuffer[i]+MinusDiBuffer[i]);
      if(div==0.00) TempBuffer[i]=0;
      else TempBuffer[i]=100*(MathAbs(PlusDiBuffer[i]-MinusDiBuffer[i])/div);
      i--;
     }
//---- ADX is exponential moving average on DX
   for(i=0; i<limit; i++)
      ADXBuffer[i]=iMAOnArray(TempBuffer,Bars,ADXPeriod,0,MODE_EMA,i);
//----
   return(0);
  }
//+------------------------------------------------------------------+
```

```
//+------------------------------------------------------------------+
//|                                                 Alligator.mq4 |
//|                      Copyright © 2004, MetaQuotes Software Corp. |
//|                                       http://www.metaquotes.net/ |
//+------------------------------------------------------------------+
#property copyright "Copyright © 2004, MetaQuotes Software Corp."
#property link      "http://www.metaquotes.net/"

#property indicator_chart_window
#property indicator_buffers 3
#property indicator_color1 Blue
#property indicator_color2 Red
#property indicator_color3 Lime
//---- input parameters
extern int JawsPeriod=13;
extern int JawsShift=8;
extern int TeethPeriod=8;
extern int TeethShift=5;
extern int LipsPeriod=5;
extern int LipsShift=3;
//---- indicator buffers
double ExtBlueBuffer[];
double ExtRedBuffer[];
double ExtLimeBuffer[];

//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int init()
  {
//---- line shifts when drawing
   SetIndexShift(0,JawsShift);
   SetIndexShift(1,TeethShift);
   SetIndexShift(2,LipsShift);
//---- first positions skipped when drawing
   SetIndexDrawBegin(0,JawsShift+JawsPeriod);
   SetIndexDrawBegin(1,TeethShift+TeethPeriod);
   SetIndexDrawBegin(2,LipsShift+LipsPeriod);
//---- 3 indicator buffers mapping
   SetIndexBuffer(0,ExtBlueBuffer);
   SetIndexBuffer(1,ExtRedBuffer);
   SetIndexBuffer(2,ExtLimeBuffer);
//---- drawing settings
   SetIndexStyle(0,DRAW_LINE);
   SetIndexStyle(1,DRAW_LINE);
   SetIndexStyle(2,DRAW_LINE);
//---- index labels
   SetIndexLabel(0,"Gator Jaws");
   SetIndexLabel(1,"Gator Teeth");
   SetIndexLabel(2,"Gator Lips");
//---- initialization done
   return(0);
  }
//+------------------------------------------------------------------+
//| Bill Williams' Alligator                                         |
//+------------------------------------------------------------------+
int start()
  {
   int limit;
   int counted_bars=IndicatorCounted();
//---- check for possible errors
   if(counted_bars<0) return(-1);
//---- last counted bar will be recounted
   if(counted_bars>0) counted_bars--;
   limit=Bars-counted_bars;
//---- main loop
   for(int i=0; i<limit; i++)
     {
      //---- ma_shift set to 0 because SetIndexShift called abowe
      ExtBlueBuffer[i]=iMA(NULL,0,JawsPeriod,0,MODE_SMMA,PRICE_MEDIAN,i);
      ExtRedBuffer[i]=iMA(NULL,0,TeethPeriod,0,MODE_SMMA,PRICE_MEDIAN,i);
      ExtLimeBuffer[i]=iMA(NULL,0,LipsPeriod,0,MODE_SMMA,PRICE_MEDIAN,i);
     }
//---- done
   return(0);
  }
//+------------------------------------------------------------------+
```

```
//+------------------------------------------------------------------+
//|                                                      Awesome.mq4 |
//|                      Copyright © 2005, MetaQuotes Software Corp. |
//|                                       http://www.metaquotes.net/ |
//+------------------------------------------------------------------+
#property  copyright "Copyright © 2005, MetaQuotes Software Corp."
#property  link      "http://www.metaquotes.net/"
//---- indicator settings
#property  indicator_separate_window
#property  indicator_buffers 3
#property  indicator_color1  Black
#property  indicator_color2  Green
#property  indicator_color3  Red

//---- indicator buffers
double     ExtBuffer0[];
double     ExtBuffer1[];
double     ExtBuffer2[];

//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int init()
  {
   //---- drawing settings
   SetIndexStyle(0,DRAW_NONE);
   SetIndexStyle(1,DRAW_HISTOGRAM);
   SetIndexStyle(2,DRAW_HISTOGRAM);
   IndicatorDigits(Digits+1);
   SetIndexDrawBegin(0,34);
   SetIndexDrawBegin(1,34);
   SetIndexDrawBegin(2,34);
//---- 3 indicator buffers mapping
   SetIndexBuffer(0,ExtBuffer0);
   SetIndexBuffer(1,ExtBuffer1);
   SetIndexBuffer(2,ExtBuffer2);
//---- name for DataWindow and indicator subwindow label
```

```
   IndicatorShortName("AO");
   SetIndexLabel(1,NULL);
   SetIndexLabel(2,NULL);
//---- initialization done
   return(0);
  }
//+------------------------------------------------------------------+
//| Awesome Oscillator                                               |
//+------------------------------------------------------------------+
int start()
  {
   int    limit;
   int    counted_bars=IndicatorCounted();
   double prev,current;
//---- last counted bar will be recounted
   if(counted_bars>0) counted_bars--;
   limit=Bars-counted_bars;
//---- macd
   for(int i=0; i<limit; i++)
      ExtBuffer0[i]=iMA(NULL,0,5,0,MODE_SMA,PRICE_MEDIAN,i)-
iMA(NULL,0,34,0,MODE_SMA,PRICE_MEDIAN,i);
//---- dispatch values between 2 buffers
   bool up=true;
   for(i=limit-1; i>=0; i--)
     {
      current=ExtBuffer0[i];
      prev=ExtBuffer0[i+1];
      if(current>prev) up=true;
      if(current<prev) up=false;
      if(!up)
        {
         ExtBuffer2[i]=current;
         ExtBuffer1[i]=0.0;
        }
      else
        {
         ExtBuffer1[i]=current;
         ExtBuffer2[i]=0.0;
        }
     }
//---- done
   return(0);
  }
```

Samples Includes

```
#import "ExpertSample.dll"
int    GetIntValue(int);
double GetDoubleValue(double);
string GetStringValue(string);
double GetArrayItemValue(double arr[],int,int);
bool   SetArrayItemValue(double& arr[],int,int,double);
double GetRatesItemValue(double rates[][6],int,int,int);
int    SortStringArray(string& arr[],int);
int    ProcessStringArray(string& arr[],int);
```

Samples Indicators

```
//+------------------------------------------------------------------+
//|                                                          ATR.mq4 |
//|                      Copyright © 2005, MetaQuotes Software Corp. |
//|                                       http://www.metaquotes.net/ |
//+------------------------------------------------------------------+
#property copyright "Copyright © 2005, MetaQuotes Software Corp."
```

```
#property link      "http://www.metaquotes.net/"

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_color1 DodgerBlue
//---- input parameters
extern int AtrPeriod=14;
//---- buffers
double AtrBuffer[];
double TempBuffer[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int init()
  {
   string short_name;
//---- 1 additional buffer used for counting.
   IndicatorBuffers(2);
//---- indicator line
   SetIndexStyle(0,DRAW_LINE);
   SetIndexBuffer(0,AtrBuffer);
   SetIndexBuffer(1,TempBuffer);
//---- name for DataWindow and indicator subwindow label
   short_name="ATR("+AtrPeriod+")";
   IndicatorShortName(short_name);
   SetIndexLabel(0,short_name);
//----
   SetIndexDrawBegin(0,AtrPeriod);
//----
   return(0);
  }
//+------------------------------------------------------------------+
//| Average True Range                                               |
//+------------------------------------------------------------------+
int start()
  {
   int i,counted_bars=IndicatorCounted();
//----
   if(Bars<=AtrPeriod) return(0);
//---- initial zero
   if(counted_bars<1)
      for(i=1;i<=AtrPeriod;i++) AtrBuffer[Bars-i]=0.0;
//----
   i=Bars-counted_bars-1;
   while(i>=0)
     {
      double high=High[i];
      double low =Low[i];
      if(i==Bars-1) TempBuffer[i]=high-low;
      else
        {
         double prevclose=Close[i+1];
         TempBuffer[i]=MathMax(high,prevclose)-MathMin(low,prevclose);
        }
      i--;
     }
//----
   if(counted_bars>0) counted_bars--;
   int limit=Bars-counted_bars;
   for(i=0; i<limit; i++)
      AtrBuffer[i]=iMAOnArray(TempBuffer,Bars,AtrPeriod,0,MODE_SMA,i);
//----
   return(0);
  }
//+------------------------------------------------------------------+
```

```
//+------------------------------------------------------------------+
//|                                                    Bands.mq4 |
//|                      Copyright © 2005, MetaQuotes Software Corp. |
//|                                       http://www.metaquotes.net/ |
//+------------------------------------------------------------------+
#property copyright "Copyright © 2005, MetaQuotes Software Corp."
#property link      "http://www.metaquotes.net/"

#property indicator_chart_window
#property indicator_buffers 3
#property indicator_color1 LightSeaGreen
#property indicator_color2 LightSeaGreen
#property indicator_color3 LightSeaGreen
//---- indicator parameters
extern int    BandsPeriod=20;
extern int    BandsShift=0;
extern double BandsDeviations=2.0;
//---- buffers
double MovingBuffer[];
double UpperBuffer[];
double LowerBuffer[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int init()
  {
//---- indicators
   SetIndexStyle(0,DRAW_LINE);
   SetIndexBuffer(0,MovingBuffer);
   SetIndexStyle(1,DRAW_LINE);
   SetIndexBuffer(1,UpperBuffer);
   SetIndexStyle(2,DRAW_LINE);
   SetIndexBuffer(2,LowerBuffer);
//----
   SetIndexDrawBegin(0,BandsPeriod+BandsShift);
   SetIndexDrawBegin(1,BandsPeriod+BandsShift);
   SetIndexDrawBegin(2,BandsPeriod+BandsShift);
//----
   return(0);
  }
//+------------------------------------------------------------------+
//| Bollinger Bands                                                  |
//+------------------------------------------------------------------+
int start()
  {
   int    i,k,counted_bars=IndicatorCounted();
   double deviation;
   double sum,oldval,newres;
//----
   if(Bars<=BandsPeriod) return(0);
//---- initial zero
   if(counted_bars<1)
      for(i=1;i<=BandsPeriod;i++)
        {
         MovingBuffer[Bars-i]=EMPTY_VALUE;
         UpperBuffer[Bars-i]=EMPTY_VALUE;
         LowerBuffer[Bars-i]=EMPTY_VALUE;
        }
//----
   int limit=Bars-counted_bars;
   if(counted_bars>0) limit++;
   for(i=0; i<limit; i++)
      MovingBuffer[i]=iMA(NULL,0,BandsPeriod,BandsShift,MODE_SMA,PRICE_CLOSE,i);
//----
   i=Bars-BandsPeriod+1;
   if(counted_bars>BandsPeriod-1) i=Bars-counted_bars-1;
   while(i>=0)
     {
      sum=0.0;
      k=i+BandsPeriod-1;
      oldval=MovingBuffer[i];
      while(k>=i)
        {
         newres=Close[k]-oldval;
         sum+=newres*newres;
         k--;
        }
      deviation=BandsDeviations*MathSqrt(sum/BandsPeriod);;
      UpperBuffer[i]=oldval+deviation;
      LowerBuffer[i]=oldval-deviation;
      i--;
     }
//----
   return(0);
  }
//+------------------------------------------------------------------+
```

```
//+------------------------------------------------------------------+
//|                                                        Bears.mq4 |
//|                      Copyright © 2005, MetaQuotes Software Corp. |
//|                                       http://www.metaquotes.net/ |
//+------------------------------------------------------------------+
#property copyright "Copyright © 2005, MetaQuotes Software Corp."
#property link      "http://www.metaquotes.net/"

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_color1 Silver
//---- input parameters
extern int BearsPeriod=13;
//---- buffers
double BearsBuffer[];
double TempBuffer[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int init()
  {
   string short_name;
//---- 1 additional buffer used for counting.
   IndicatorBuffers(2);
   IndicatorDigits(Digits);
//---- indicator line
   SetIndexStyle(0,DRAW_HISTOGRAM);
   SetIndexBuffer(0,BearsBuffer);
   SetIndexBuffer(1,TempBuffer);
//---- name for DataWindow and indicator subwindow label
   short_name="Bears("+BearsPeriod+")";
   IndicatorShortName(short_name);
   SetIndexLabel(0,short_name);
//----
   return(0);
  }
//+------------------------------------------------------------------+
//| Bears Power                                                      |
//+------------------------------------------------------------------+
int start()
  {
   int i,counted_bars=IndicatorCounted();
//----
   if(Bars<=BearsPeriod) return(0);
//----
   int limit=Bars-counted_bars;
   if(counted_bars>0) limit++;
   for(i=0; i<limit; i++)
```

```
      TempBuffer[i]=iMA(NULL,0,BearsPeriod,0,MODE_EMA,PRICE_CLOSE,i);
//----
   i=Bars-counted_bars-1;
   while(i>=0)
     {
      BearsBuffer[i]=Low[i]-TempBuffer[i];
      i--;
     }
//----
   return(0);
  }
//+------------------------------------------------------------------+
```

```
//+------------------------------------------------------------------+
//|                                                        Bulls.mq4 |
//|                      Copyright © 2005, MetaQuotes Software Corp. |
//|                                       http://www.metaquotes.net/ |
//+------------------------------------------------------------------+
#property copyright "Copyright © 2005, MetaQuotes Software Corp."
#property link      "http://www.metaquotes.net/"

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_color1 Silver
//---- input parameters
extern int BullsPeriod=13;
//---- buffers
double BullsBuffer[];
double TempBuffer[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int init()
  {
   string short_name;
//---- 1 additional buffer used for counting.
   IndicatorBuffers(2);
   IndicatorDigits(Digits);
//---- indicator line
   SetIndexStyle(0,DRAW_HISTOGRAM);
   SetIndexBuffer(0,BullsBuffer);
   SetIndexBuffer(1,TempBuffer);
//---- name for DataWindow and indicator subwindow label
   short_name="Bulls("+BullsPeriod+")";
   IndicatorShortName(short_name);
   SetIndexLabel(0,short_name);
//----
   return(0);
  }
//+------------------------------------------------------------------+
//| Bulls Power                                                      |
//+------------------------------------------------------------------+
int start()
  {
   int i,counted_bars=IndicatorCounted();
//----
   if(Bars<=BullsPeriod) return(0);
//----
   int limit=Bars-counted_bars;
   if(counted_bars>0) limit++;
   for(i=0; i<limit; i++)
      TempBuffer[i]=iMA(NULL,0,BullsPeriod,0,MODE_EMA,PRICE_CLOSE,i);
//----
   i=Bars-counted_bars-1;
   while(i>=0)
     {
      BullsBuffer[i]=High[i]-TempBuffer[i];
      i--;
     }
//----
   return(0);
  }
//+------------------------------------------------------------------+
```

```
//+------------------------------------------------------------------+
//|                                                 Heiken Ashi.mq4 |
//|                      Copyright c 2004, MetaQuotes Software Corp. |
//|                                       http://www.metaquotes.net |
//+------------------------------------------------------------------+
//| For Heiken Ashi we recommend next chart settings ( press F8 or   |
//| select on menu 'Charts'->'Properties...'):                       |
//|  - On 'Color' Tab select 'Black' for 'Line Graph'                |
//|  - On 'Common' Tab disable 'Chart on Foreground' checkbox and    |
//|    select 'Line Chart' radiobutton                               |
//+------------------------------------------------------------------+
#property copyright "Copyright © 2004, MetaQuotes Software Corp."
#property link      "http://www.metaquotes.net"

#property indicator_chart_window
#property indicator_buffers 4
#property indicator_color1 Red
#property indicator_color2 White
#property indicator_color3 Red
#property indicator_color4 White
//---- buffers
double ExtMapBuffer1[];
double ExtMapBuffer2[];
double ExtMapBuffer3[];
double ExtMapBuffer4[];
//----
int ExtCountedBars=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//|------------------------------------------------------------------|
int init()
  {
//---- indicators
   SetIndexStyle(0,DRAW_HISTOGRAM, 0, 1, Red);
   SetIndexBuffer(0, ExtMapBuffer1);
   SetIndexStyle(1,DRAW_HISTOGRAM, 0, 1, White);
   SetIndexBuffer(1, ExtMapBuffer2);
   SetIndexStyle(2,DRAW_HISTOGRAM, 0, 3, Red);
   SetIndexBuffer(2, ExtMapBuffer3);
   SetIndexStyle(3,DRAW_HISTOGRAM, 0, 3, White);
   SetIndexBuffer(3, ExtMapBuffer4);
//----
   SetIndexDrawBegin(0,10);
   SetIndexDrawBegin(1,10);
   SetIndexDrawBegin(2,10);
   SetIndexDrawBegin(3,10);
//---- indicator buffers mapping
   SetIndexBuffer(0,ExtMapBuffer1);
   SetIndexBuffer(1,ExtMapBuffer2);
   SetIndexBuffer(2,ExtMapBuffer3);
   SetIndexBuffer(3,ExtMapBuffer4);
//---- initialization done
   return(0);
  }
//+------------------------------------------------------------------+
//| Custom indicator deinitialization function                       |
//+------------------------------------------------------------------+
int deinit()
  {
//---- TODO: add your code here
//----
   return(0);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int start()
  {
   double haOpen, haHigh, haLow, haClose;
   if(Bars<=10) return(0);
   ExtCountedBars=IndicatorCounted();
//---- check for possible errors
   if (ExtCountedBars<0) return(-1);
//---- last counted bar will be recounted
   if (ExtCountedBars>0) ExtCountedBars--;
   int pos=Bars-ExtCountedBars-1;
   while(pos>=0)
     {
      haOpen=(ExtMapBuffer3[pos+1]+ExtMapBuffer4[pos+1])/2;
      haClose=(Open[pos]+High[pos]+Low[pos]+Close[pos])/4;
      haHigh=MathMax(High[pos], MathMax(haOpen, haClose));
      haLow=MathMin(Low[pos], MathMin(haOpen, haClose));
      if (haOpen<haClose)
        {
         ExtMapBuffer1[pos]=haLow;
         ExtMapBuffer2[pos]=haHigh;
        }
      else
        {
         ExtMapBuffer1[pos]=haHigh;
         ExtMapBuffer2[pos]=haLow;
        }
      ExtMapBuffer3[pos]=haOpen;
      ExtMapBuffer4[pos]=haClose;
         pos--;
     }
//----
   return(0);
  }
//+------------------------------------------------------------------+
```

Samples Scripts

```
//+------------------------------------------------------------------+
//|                                                         close.mq4 |
//|                      Copyright © 2004, MetaQuotes Software Corp. |
//|                                       http://www.metaquotes.net/ |
//+------------------------------------------------------------------+
#property copyright "Copyright © 2004, MetaQuotes Software Corp."
#property link      "http://www.metaquotes.net/"
#property show_confirm

//+------------------------------------------------------------------+
//| script "close first market order if it is first in the list"     |
//+------------------------------------------------------------------+
int start()
  {
   bool   result;
   double price;
   int    cmd,error;
//----
   if(OrderSelect(0,SELECT_BY_POS,MODE_TRADES))
     {
      cmd=OrderType();
      //---- first order is buy or sell
      if(cmd==OP_BUY || cmd==OP_SELL)
        {
         while(true)
           {
            if(cmd==OP_BUY) price=Bid;
            else            price=Ask;
            result=OrderClose(OrderTicket(),OrderLots(),price,3,CLR_NONE);
            if(result!=TRUE) { error=GetLastError(); Print("LastError = ",error); }
            else error=0;
            if(error==135) RefreshRates();
            else break;
           }
        }
     }
   else Print( "Error when order select ", GetLastError());
//----
   return(0);
  }
//+------------------------------------------------------------------+
```

```
//+------------------------------------------------------------------+
//|                                               delete_pending.mq4 |
//|                      Copyright © 2004, MetaQuotes Software Corp. |
//|                                       http://www.metaquotes.net/ |
//+------------------------------------------------------------------+
#property copyright "Copyright © 2004, MetaQuotes Software Corp."
#property link      "http://www.metaquotes.net/"
#property show_confirm

//+------------------------------------------------------------------+
//| script "delete first pending order"                              |
//+------------------------------------------------------------------+
int start()
  {
   bool   result;
   int    cmd,total;
//----
   total=OrdersTotal();
//----
   for(int i=0; i<total; i++)
     {
      if(OrderSelect(i,SELECT_BY_POS,MODE_TRADES))
        {
         cmd=OrderType();
         //---- pending orders only are considered
         if(cmd!=OP_BUY && cmd!=OP_SELL)
           {
            //---- print selected order
            OrderPrint();
            //---- delete first pending order
            result=OrderDelete(OrderTicket());
            if(result!=TRUE) Print("LastError = ", GetLastError());
            break;
           }
        }
      else { Print( "Error when order select ", GetLastError()); break; }
     }
//----
   return(0);
  }
//+------------------------------------------------------------------+
```

```
//+------------------------------------------------------------------+
//|                                                       modify.mq4 |
//|                      Copyright © 2004, MetaQuotes Software Corp. |
//|                                       http://www.metaquotes.net/ |
//+------------------------------------------------------------------+
#property copyright "Copyright © 2004, MetaQuotes Software Corp."
#property link      "http://www.metaquotes.net/"
#property show_confirm

//+------------------------------------------------------------------+
//| script "modify first market order"                               |
//+------------------------------------------------------------------+
int start()
  {
   bool   result;
   double stop_loss,point;
   int    cmd,total;
//----
   total=OrdersTotal();
   point=MarketInfo(Symbol(),MODE_POINT);
//----
   for(int i=0; i<total; i++)
     {
      if(OrderSelect(i,SELECT_BY_POS,MODE_TRADES))
        {
         //---- print selected order
         OrderPrint();
         cmd=OrderType();
         //---- buy or sell orders are considered
         if(cmd==OP_BUY || cmd==OP_SELL)
           {
            //---- modify first market order
            while(true)
              {
               if(cmd==OP_BUY) stop_loss=Bid-20*point;
               else            stop_loss=Ask+20*point;
               result=OrderModify(OrderTicket(),0,stop_loss,0,0,CLR_NONE);
               if(result!=TRUE) Print("LastError = ", GetLastError());
               if(result==135) RefreshRates();
               else break;
              }
             //---- print modified order (it still selected after modify)
             OrderPrint();
             break;
           }
        }
      else { Print( "Error when order select ", GetLastError()); break; }
     }
//----
   return(0);
  }
//+------------------------------------------------------------------+
```

```
//+------------------------------------------------------------------+
//|                                               modify_pending.mq4 |
//|                      Copyright © 2004, MetaQuotes Software Corp. |
//|                                       http://www.metaquotes.net/ |
//+------------------------------------------------------------------+
#property copyright "Copyright © 2004, MetaQuotes Software Corp."
#property link      "http://www.metaquotes.net/"
#property show_confirm

//+------------------------------------------------------------------+
//| script "modify first pending order"                              |
//+------------------------------------------------------------------+
int start()
  {
   bool   result;
   double price,point;
   int    cmd,total;
   int    expiration;
//----
```

```
   total=OrdersTotal();
   point=MarketInfo(Symbol(),MODE_POINT);
//----
   for(int i=0; i<total; i++)
     {
      if(OrderSelect(i,SELECT_BY_POS,MODE_TRADES))
        {
         cmd=OrderType();
         //---- pending orders only are considered
         if(cmd!=OP_BUY && cmd!=OP_SELL)
           {
            //---- print selected order
            OrderPrint();
            //---- modify first pending order
            price=OrderOpenPrice()-10*point;
            expiration=OrderExpiration();
            result=OrderModify(OrderTicket(),price,0,0,expiration,CLR_NONE);
            if(result!=TRUE) Print("LastError = ", GetLastError());
            //---- print modified order (it still selected after modify)
            else OrderPrint();
            break;
           }
        }
      else { Print( "Error when order select ", GetLastError()); break; }
     }
//----
   return(0);
  }
//+------------------------------------------------------------------+
```

```
//+------------------------------------------------------------------+
//|                                                 rotate_text.mq4 |
//|                      Copyright © 2004, MetaQuotes Software Corp. |
//|                                       http://www.metaquotes.net/ |
//+------------------------------------------------------------------+
#property copyright "Copyright © 2004, MetaQuotes Software Corp."
#property link      "http://www.metaquotes.net/"
#include <stdlib.mqh>

string line_name="rotating_line";
string object_name1="rotating_text";

void init()
  {
   Print("point = ", Point,"   bars=",Bars);
  }
//+------------------------------------------------------------------+
//|                                                                  |
//+------------------------------------------------------------------+
void deinit()
  {
   ObjectDelete(line_name);
   ObjectDelete(object_name1);
   ObjectsRedraw();
  }
//+------------------------------------------------------------------+
//| script program start function                                    |
//+------------------------------------------------------------------+
int start()
  {
   int    time2;
   int    error,index,fontsize=10;
   double price,price1,price2;
   double angle=0.0;
//----
   price2=High[10]+Point*10;
   ObjectCreate(line_name, OBJ_TRENDBYANGLE, 0, Time[10], price2);
   index=20;
   ObjectCreate(object_name1, OBJ_TEXT, 0, Time[index], Low[index]-Point*100);
   ObjectSetText(object_name1, "rotating_text", fontsize);
   while(IsStopped()==false)
     {
      index++;
      price=ObjectGet(object_name1, OBJPROP_PRICE1)+Point;
      error=GetLastError();
      if(error!=0)
        {
         Print(object_name1," : ",ErrorDescription(error));
         break;
        }
      ObjectMove(object_name1, 0, Time[index], price);
      ObjectSet(object_name1, OBJPROP_ANGLE, angle*2);
      ObjectSet(object_name1, OBJPROP_FONTSIZE, fontsize);
      ObjectSet(line_name, OBJPROP_WIDTH, angle/18.0);
      double line_angle=360.0-angle;
      if(line_angle==90.0)  ObjectSet(line_name, OBJPROP_PRICE2, price2+Point*50);
      if(line_angle==270.0) ObjectSet(line_name, OBJPROP_PRICE2, price2-Point*50);
      time2=ObjectGet(line_name,OBJPROP_TIME2);
      if(line_angle>90.0 && line_angle<270.0) time2=Time[index+10];
      else                                    time2=Time[0];
      ObjectSet(line_name, OBJPROP_TIME2, time2);
      ObjectSet(line_name, OBJPROP_ANGLE, line_angle);
      ObjectsRedraw();
      angle+=3.0;
      if(angle>=360.0) angle=360.0-angle;
      fontsize++;
      if(fontsize>48) fontsize=6;
      Sleep(500);
      price1=ObjectGetValueByShift(line_name, index);
      if(GetLastError()==0)
        {
         if(MathAbs(price1-price) < Point*50)
           {
            Print("price=",price,"  price1=", price1);
            ObjectSetText(object_name1, "REMOVED", 48, "Arial", RGB(255,215,0));
            ObjectsRedraw();
            Sleep(5000);
//            ObjectDelete(object_name1);
           }
        }
     }
   return(0);
  }
//+------------------------------------------------------------------+
```

```
//+------------------------------------------------------------------+
//|                                                  send_pending.mq4 |
//|                      Copyright © 2004, MetaQuotes Software Corp. |
//|                                       http://www.metaquotes.net/ |
//+------------------------------------------------------------------+
#property copyright "Copyright © 2004, MetaQuotes Software Corp."
#property link      "http://www.metaquotes.net/"
#property show_confirm

//+------------------------------------------------------------------+
//| script "send pending order with expiration data"                 |
//+------------------------------------------------------------------+
int start()
  {
   int    ticket,expiration;
   double point;
//----
   point=MarketInfo(Symbol(),MODE_POINT);
   expiration=CurTime()+PERIOD_D1*60;
//----
   while(true)
     {
      ticket=OrderSend(Symbol(),OP_SELLSTOP,1.0,Bid-100*point,0,0,0,"some
comment",16384,expiration,Green);
      if(ticket<=0) Print("Error = ",GetLastError());
      else { Print("ticket = ",ticket); break; }
      //---- 10 seconds wait
      Sleep(10000);
     }
//----
   return(0);
  }
//+------------------------------------------------------------------+
```

```
//+------------------------------------------------------------------+
//|                                                     trade.mq4 |
//|                      Copyright © 2004, MetaQuotes Software Corp. |
//|                                       http://www.metaquotes.net/ |
//+------------------------------------------------------------------+
#property copyright "Copyright © 2004, MetaQuotes Software Corp."
#property link      "http://www.metaquotes.net/"

#include <stdlib.mqh>
#include <WinUser32.mqh>
//+------------------------------------------------------------------+
//| script "trading for all money"                                   |
//+------------------------------------------------------------------+
int start()
  {
//----
   if(MessageBox("Do you really want to BUY 1.00 "+Symbol()+" at ASK price?    ",
                 "Script",MB_YESNO|MB_ICONQUESTION)!=IDYES) return(1);
//----
   int ticket=OrderSend(Symbol(),OP_BUY,1.0,Ask,3,0,0,"expert comment",255,0,CLR_NONE);
   if(ticket<1)
     {
      int error=GetLastError();
      Print("Error = ",ErrorDescription(error));
      return;
     }
//----
   OrderPrint();
   return(0);
  }
//+------------------------------------------------------------------+
```

Samples Program

```
//+------------------------------------------------------------------+
//|                                           ExportFunctions.mq4 |
//|                      Copyright © 2005, MetaQuotes Software Corp. |
//|                                       http://www.metaquotes.net/ |
//+------------------------------------------------------------------+
#property copyright "Copyright © 2005, MetaQuotes Software Corp."
#property link      "http://www.metaquotes.net/"
#include <sampledll.mqh>

#define TIME_INDEX   0
#define OPEN_INDEX   1
#define LOW_INDEX    2
#define HIGH_INDEX   3
#define CLOSE_INDEX  4
#define VOLUME_INDEX 5
//+------------------------------------------------------------------+
//| expert initialization function                                   |
//+------------------------------------------------------------------+
int init()
  {
   double ret,some_value=10.5;
   string sret;
   int    cnt;
   string strarray[6]={ "first", "second", "third", "fourth", "fifth" };
//---- simple dll-functions call
   cnt=GetIntValue(some_value);
   Print("Returned value is ",cnt);
   ret=GetDoubleValue(some_value);
   Print("Returned value is ",ret);
   sret=GetStringValue("some string");
   Print("Returned value is ",sret);
//----
   cnt=SortStringArray(strarray,ArraySize(strarray));
   for(int i=0; i<cnt; i++) Print(i," - ",strarray[i]);
   cnt=ProcessStringArray(strarray,ArraySize(strarray));
   for(i=0; i<cnt; i++) Print(i," - ",strarray[i]);
//----
   return(0);
  }
//+------------------------------------------------------------------+
//| array functions call                                             |
//+------------------------------------------------------------------+
int start()
  {
   double price;
   double arr[5]={1.5, 2.6, 3.7, 4.8, 5.9 };
   double rates[][6];
//---- get first item from passed array
   price=GetArrayItemValue(arr,5,0);
   Print("Returned from arr[0] ",price);
//---- change second item in the passed array
   if(SetArrayItemValue(arr,5,1,1234.5)==true)
      Print("Changed to ",arr[1]);
//---- get current close
   ArrayCopyRates(rates);
   price=GetRatesItemValue(rates,Bars,0,CLOSE_INDEX);
   Print("Returned from Close ",price);
//----
   return(0);
  }
//+------------------------------------------------------------------+
```

```
//+------------------------------------------------------------------+
//|                                                  MACD Sample.mq4 |
//|                      Copyright © 2005, MetaQuotes Software Corp. |
//|                                       http://www.metaquotes.net/ |
//+------------------------------------------------------------------+

extern double TakeProfit = 50;
extern double Lots = 0.1;
extern double TrailingStop = 30;
extern double MACDOpenLevel=3;
extern double MACDCloseLevel=2;
extern double MATrendPeriod=26;

//+------------------------------------------------------------------+
//|                                                                  |
//+------------------------------------------------------------------+
int start()
  {
   double MacdCurrent, MacdPrevious, SignalCurrent;
   double SignalPrevious, MaCurrent, MaPrevious;
   int cnt, ticket, total;
```

```
// initial data checks
// it is important to make sure that the expert works with a normal
// chart and the user did not make any mistakes setting external
// variables (Lots, StopLoss, TakeProfit,
// TrailingStop) in our case, we check TakeProfit
// on a chart of less than 100 bars
   if(Bars<100)
     {
      Print("bars less than 100");
      return(0);
     }
   if(TakeProfit<10)
     {
      Print("TakeProfit less than 10");
      return(0);  // check TakeProfit
     }
// to simplify the coding and speed up access
// data are put into internal variables
   MacdCurrent=iMACD(NULL,0,12,26,9,PRICE_CLOSE,MODE_MAIN,0);
   MacdPrevious=iMACD(NULL,0,12,26,9,PRICE_CLOSE,MODE_MAIN,1);
   SignalCurrent=iMACD(NULL,0,12,26,9,PRICE_CLOSE,MODE_SIGNAL,0);
   SignalPrevious=iMACD(NULL,0,12,26,9,PRICE_CLOSE,MODE_SIGNAL,1);
   MaCurrent=iMA(NULL,0,MATrendPeriod,0,MODE_EMA,PRICE_CLOSE,0);
   MaPrevious=iMA(NULL,0,MATrendPeriod,0,MODE_EMA,PRICE_CLOSE,1);

   total=OrdersTotal();
   if(total<1)
     {
      // no opened orders identified
      if(AccountFreeMargin()<(1000*Lots))
        {
         Print("We have no money. Free Margin = ", AccountFreeMargin());
         return(0);
        }
      // check for long position (BUY) possibility
      if(MacdCurrent<0 && MacdCurrent>SignalCurrent && MacdPrevious<SignalPrevious &&
         MathAbs(MacdCurrent)>(MACDOpenLevel*Point) && MaCurrent>MaPrevious)
        {
         ticket=OrderSend(Symbol(),OP_BUY,Lots,Ask,3,0,Ask+TakeProfit*Point,"macd
sample",16384,0,Green);
         if(ticket>0)
           {
            if(OrderSelect(ticket,SELECT_BY_TICKET,MODE_TRADES)) Print("BUY order
opened : ",OrderOpenPrice());
           }
         else Print("Error opening BUY order : ",GetLastError());
         return(0);
        }
      // check for short position (SELL) possibility
      if(MacdCurrent>0 && MacdCurrent<SignalCurrent && MacdPrevious>SignalPrevious &&
         MacdCurrent>(MACDOpenLevel*Point) && MaCurrent<MaPrevious)
        {
         ticket=OrderSend(Symbol(),OP_SELL,Lots,Bid,3,0,Bid-TakeProfit*Point,"macd
sample",16384,0,Red);
         if(ticket>0)
           {
            if(OrderSelect(ticket,SELECT_BY_TICKET,MODE_TRADES)) Print("SELL order
opened : ",OrderOpenPrice());
           }
         else Print("Error opening SELL order : ",GetLastError());
         return(0);
        }
      return(0);
     }
   // it is important to enter the market correctly,
   // but it is more important to exit it correctly...
   for(cnt=0;cnt<total;cnt++)
```

```
      {
       OrderSelect(cnt, SELECT_BY_POS, MODE_TRADES);
       if(OrderType()<=OP_SELL &&   // check for opened position
          OrderSymbol()==Symbol())  // check for symbol
         {
          if(OrderType()==OP_BUY)   // long position is opened
            {
             // should it be closed?
             if(MacdCurrent>0 && MacdCurrent<SignalCurrent &&
MacdPrevious>SignalPrevious &&
                MacdCurrent>(MACDCloseLevel*Point))
                 {
                  OrderClose(OrderTicket(),OrderLots(),Bid,3,Violet); // close position
                  return(0); // exit
                 }
             // check for trailing stop
             if(TrailingStop>0)
               {
                if(Bid-OrderOpenPrice()>Point*TrailingStop)
                  {
                   if(OrderStopLoss()<Bid-Point*TrailingStop)
                     {
                      OrderModify(OrderTicket(),OrderOpenPrice(),Bid-
Point*TrailingStop,OrderTakeProfit(),0,Green);
                      return(0);
                     }
                  }
               }
            }
          else // go to short position
            {
             // should it be closed?
             if(MacdCurrent<0 && MacdCurrent>SignalCurrent &&
                MacdPrevious<SignalPrevious &&
MathAbs(MacdCurrent)>(MACDCloseLevel*Point))
               {
                OrderClose(OrderTicket(),OrderLots(),Ask,3,Violet); // close position
                return(0); // exit
               }
             // check for trailing stop
             if(TrailingStop>0)
               {
                if((OrderOpenPrice()-Ask)>(Point*TrailingStop))
                  {
                   if((OrderStopLoss()>(Ask+Point*TrailingStop)) ||
(OrderStopLoss()==0))
                     {
                      OrderModify(OrderTicket(),OrderOpenPrice(),Ask+Point*TrailingStop,
OrderTakeProfit(),0,Red);
                      return(0);
                     }
                  }
               }
            }
         }
      }
   return(0);
  }
// the end.
```

```
//+------------------------------------------------------------------+
//|                                               Moving Average.mq4 |
//|                      Copyright © 2005, MetaQuotes Software Corp. |
//|                                       http://www.metaquotes.net/ |
//+------------------------------------------------------------------+
#define MAGICMA  20050610
```

```
extern double Lots               = 0.1;
extern double MaximumRisk        = 0.02;
extern double DecreaseFactor     = 3;
extern double MovingPeriod       = 12;
extern double MovingShift        = 6;
//+------------------------------------------------------------------+
//| Calculate open positions                                         |
//+------------------------------------------------------------------+
int CalculateCurrentOrders(string symbol)
  {
   int buys=0,sells=0;
//----
   for(int i=0;i<OrdersTotal();i++)
     {
      if(OrderSelect(i,SELECT_BY_POS,MODE_TRADES)==false) break;
      if(OrderSymbol()==Symbol() && OrderMagicNumber()==MAGICMA)
        {
         if(OrderType()==OP_BUY)  buys++;
         if(OrderType()==OP_SELL) sells++;
        }
     }
//---- return orders volume
   if(buys>0) return(buys);
   else       return(-sells);
  }
//+------------------------------------------------------------------+
//| Calculate optimal lot size                                       |
//+------------------------------------------------------------------+
double LotsOptimized()
  {
   double lot=Lots;
   int    orders=HistoryTotal();     // history orders total
   int    losses=0;                  // number of losses orders without a break
//---- select lot size
   lot=NormalizeDouble(AccountFreeMargin()*MaximumRisk/1000.0,1);
//---- calcuulate number of losses orders without a break
   if(DecreaseFactor>0)
     {
      for(int i=orders-1;i>=0;i--)
        {
         if(OrderSelect(i,SELECT_BY_POS,MODE_HISTORY)==false) { Print("Error in 
history!"); break; }
         if(OrderSymbol()!=Symbol() || OrderType()>OP_SELL) continue;
         //----
         if(OrderProfit()>0) break;
         if(OrderProfit()<0) losses++;
        }
      if(losses>1) lot=NormalizeDouble(lot-lot*losses/DecreaseFactor,1);
     }
//---- return lot size
   if(lot<0.1) lot=0.1;
   return(lot);
  }
//+------------------------------------------------------------------+
//| Check for open order conditions                                  |
//+------------------------------------------------------------------+
void CheckForOpen()
  {
   double ma;
   int    res;
//---- go trading only for first tiks of new bar
   if(Volume[0]>1) return;
//---- get Moving Average
   ma=iMA(NULL,0,MovingPeriod,MovingShift,MODE_SMA,PRICE_CLOSE,0);
//---- sell conditions
   if(Open[1]>ma && Close[1]<ma)
     {
      res=OrderSend(Symbol(),OP_SELL,LotsOptimized(),Bid,3,0,0,"",MAGICMA,0,Red);
      return;
     }
//---- buy conditions
   if(Open[1]<ma && Close[1]>ma)
     {
      res=OrderSend(Symbol(),OP_BUY,LotsOptimized(),Ask,3,0,0,"",MAGICMA,0,Blue);
      return;
     }
//----
  }
//+------------------------------------------------------------------+
//| Check for close order conditions                                 |
//+------------------------------------------------------------------+
void CheckForClose()
  {
   double ma;
//---- go trading only for first tiks of new bar
   if(Volume[0]>1) return;
//---- get Moving Average
   ma=iMA(NULL,0,MovingPeriod,MovingShift,MODE_SMA,PRICE_CLOSE,0);
//----
   for(int i=0;i<OrdersTotal();i++)
     {
      if(OrderSelect(i,SELECT_BY_POS,MODE_TRADES)==false)        break;
      if(OrderMagicNumber()!=MAGICMA || OrderSymbol()!=Symbol()) continue;
      //---- check order type
      if(OrderType()==OP_BUY)
        {
         if(Open[1]>ma && Close[1]<ma)
OrderClose(OrderTicket(),OrderLots(),Bid,3,White);
         break;
        }
      if(OrderType()==OP_SELL)
        {
         if(Open[1]<ma && Close[1]>ma)
OrderClose(OrderTicket(),OrderLots(),Ask,3,White);
         break;
        }
     }
//----
  }
//+------------------------------------------------------------------+
//| Start function                                                   |
//+------------------------------------------------------------------+
void start()
  {
//---- check for history and trading
   if(Bars<100 || IsTradeAllowed()==false) return;
//---- calculate open orders by current symbol
   if(CalculateCurrentOrders(Symbol())==0) CheckForOpen();
   else                                    CheckForClose();
//----
  }
//+------------------------------------------------------------------+
```